# MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

## Génération de modèles et extraction de qualités pour le développement logiciel Agile

Georis, François

*Award date:*
2019

*Awarding institution:*
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

# Génération de modèles et extraction de qualités pour le développement logiciel Agile

François Georis

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2018–2019

# Génération de modèles et extraction de qualités pour le développement logiciel Agile

François Georis


Maître de stage : Fabian Gilson, University of Canterbury, NZ

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Pierre-Yves Schobbens

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

# RÉSUMÉ

En développement logiciel Agile, on utilise habituellement un backlog composé d'un ensemble d'histoires d'utilisateur pour documenter les exigences utilisateurs d'un projet. En règle générale, chacune d'elles correspond à une phrase assez courte respectant le template de Cohn(*"**As a** <acteur>, **I want** <objectif> ,[**so that** <bénéfice>]"*) et leur nombre est assez élevé. Cela rend très compliqué le maintien du backlog, la compréhension du domaine couvert sans une analyse détaillée et l'identification de l'impact de chaque histoire d'utilisateur sur les décisions de design. Les attributs de qualité sont l'un des facteurs les plus importants dans la prise de décisions design. Dans ce mémoire, nous proposons: (a) une technique d'identification des attributs de qualité liés à une histoire d'utilisateur et (b) une approche permettant de modéliser automatiquement les histoires d'utilisateur sous forme de diagrammes de robustesse (visualisation semi-formelle modélisant le flux d'un scénario en incluant les acteurs, les entités du domaine et les interfaces utilisateurs). Pour évaluer nos travaux, nous avons utilisé une base de données composée de 1,675 histoires d'utilisateur réparties entre 22 backlogs de projets industriels et universitaires. (a) Pour cette première technique, nous avons testé différents modèles de classification d'apprentissage automatique pour prouver qu'il est possible de détecter les histoires d'utilisateur faisant référence à un attribut de qualité et d'en identifier le type. Nos expérimentations ont pu montrer la faisabilité de notre idée avec un F1-Score de 0.71 pour la détection et un F1-Score de 0.63 pour l'identification. (b) La seconde technique permet de fusionner les diagrammes de robustesse de plusieurs histoires d'utilisateur afin d'offrir une vue d'ensemble du domaine et permettre l'identification de la redondance et des incohérences potentielles. De plus, pour faciliter l'analyse en détail d'un élément de ces diagrammes unifiés, nous proposons une approche par *"Point-De-Vue"*. Elle consiste à sélectionner un élément et à représenter seulement les objets du diagramme en lien avec celui-ci. Nous avons évalué la qualité syntaxique des diagrammes de robustesse générés en fonction du fait que l'ensemble des objets qui le composent sont connectés ou ne le sont pas.

# ABSTRACT

In Agile software development, a backlog composed of a set of user stories is usually used to document the user requirements of a project. As a general rule, each of them corresponds to a rather short sentence respecting Cohn's template (*"**As a** <actor>, **I want** <goal> ,[**so that** <benefit>]"*) and their number is quite high. This makes it very difficult to maintain the backlog, understand the area covered without a detailed analysis and identify the impact of each user story on design decisions. Quality attributes are one of the most important factors in design decision-making. In this thesis, we propose : (a) a technique for identifying quality attributes related to a user story and (b) an approach for automatically modeling user stories in the form of robustness diagrams (semi-formal visualization that models the flow of a scenario including actors, domain entities and user interfaces). To evaluate our work, a database of 1,675 user stories from 22 industrial and academic project backlogs was used. (a) For the first technique, different machine learning classification models were tested in order to prove that it is possible to detect user stories referring to a quality attribute and identify its type. The experiments have shown the feasibility of the idea with a *F1-Score* of 0.71 for detection and a *F1-Score* of 0.63 for identification. (b) The second technique allows to merge the robustness diagrams of several user histories to provide an overview of the domain and identify potential redundancy and inconsistencies. In addition, to facilitate the detailed analysis of an element of these unified diagrams, a *"view-based"* approach is proposed. It consists in selecting an element and representing only the objects of the diagram related to this object. The syntactic quality of the generated robustness diagrams were evaluated based on whether or not all of its component objects are connected.

# Remerciements

Je tiens à adresser mes plus vifs remerciements à toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé pour la rédaction de ce mémoire.

Je voudrais tout d'abord remercier mon Maître de Stage, Docteur Fabian GILSON, qui m'a accueilli à l'Université de Canterbury à Christchurch en Nouvelle-Zélande. J'ai pu, grâce à son précieux appui et à ses conseils judicieux, effectuer un stage riche tant par le sujet des travaux réalisés que par les contacts humains que j'ai pu nouer dans ce pays extraordinaire.

Je garderai aussi toujours une pensée pour les nombreuses victimes et toutes les familles touchées par l'acte terroriste qui a eu lieu dans cette belle ville durant mon séjour.

Ma reconnaissance va également au Professeur Pierre-Yves SCHOBBENS qui a accepté d'être mon promoteur dans la rédaction de ce mémoire. Je le remercie chaleureusement pour son soutien et son aide durant la préparation de ce travail.

Enfin, je souhaite témoigner ma profonde gratitude aux professeurs, aux assistants et au personnel administratif de la Faculté d'Informatique de Namur qui m'ont encadré durant ces cinq années pour me permettre d'apprendre et d'évoluer.

# CONTENTS

# INTRODUCTION

## Contexte et Problèmes

Depuis son apparition dans les années 2000, la technique de développement logiciel Agile est devenue un incontournable de l'industrie à tel point que, selon plusieurs recherches, elle a surpassé l'approche "classique" Waterfall [Dybå and Dingsøyr, 2008; Kassab, 2014; Stavru, 2014]. Parmi les différents outils proposés par Agile, la technique des histoires d'utilisateur (user stories) a été largement adoptée par les développeurs [Schön et al., 2017]. Celle-ci consiste en la rédaction des exigences d'un projet en langage naturel par les développeurs et/ou les clients. Parmi tous les templates d'histoire d'utilisateur existants, le plus utilisé en industrie est celui proposé par Cohn (ressemble étroitement à d'autres [Wautelet et al., 2014a]). Celui-ci prend la forme *"**As a** <acteur>, **I want** <objectif> ,[**so that** <bénéfice>]"* [Cohn, 2004] et il est souvent associé à des règles de bonne écriture comme le Framework INVEST [Wake, 2003] ou le Framework Quality User [Lucassen et al., 2016a]. Elles seront parfois complétées par des critères d'acceptance qui viendront ajouter des détails importants [Hoda and Murugesan, 2016].

La force des histoires d'utilisateur se manifeste dans la simplicité d'écriture tout en offrant un niveau suffisant de détails. Le fait qu'elles soient écrites en langage naturel utilisant un vocabulaire simple compris par les développeurs et les utilisateurs permet de s'assurer de la bonne compréhension des exigences fonctionnelles et non fonctionnelles définies par l'ensemble des personnes concernées. Lucassen a pu identifier que celles-ci permettent d'améliorer la productivité et la qualité du travail fourni [Lucassen et al., 2016b]. Cependant, une mauvaise rédaction causée par un manque de respect des règles de bonne écriture peut engendrer de nouveaux challenges dans la méthode de développement Agile tels qu'une sous-documentation des exigences, l'oubli de certaines exigences non fonctionnelles [Schön et al., 2017], une repriorisation constante des exigences [Inayat et al., 2015], une mauvaise vue d'ensemble du projet [Schön et al., 2017].

Plus le nombre d'histoires d'utilisateur composant le backlog d'un projet est important, plus la tâche d'analyse et de maintenance ,*e.g.,* identifier les redondances et les liens entre différentes US, ou détecter des inconsistances (mauvaise terminologie, etc.) entre les histoires d'utilisateur lorsqu'une nouvelle est ajoutée, devient compliquée pour les architectes logiciels. Il est important pour eux d'avoir dès le début d'un projet, une vue d'ensemble afin de prendre les meilleures [Schön et al., 2017] décisions de design architectural *e.g.,* choix des patterns architecturaux, stratégie ... [Bass et al., 2002]. De mauvaises décisions prises au début du développement peuvent causer de grosses modifications plus tard.

Pour avoir une vue d'ensemble correcte, les architectes doivent identifier les dépendances partagées entre les différentes histoires d'utilisateur et les sous-ensembles composant les exigences liées à une activité d'utilisateur majeure. Ces tâches sont souvent ardues vu le nombre important d'histoires d'utilisateur composant le backlog.

Les attributs de qualité tels que la performance, la sécurité, la compatibilité, l'utilisabilité, la fiabilité, la maintenabilité ou la portabilité, constituent l'un des facteurs impactant grandement les décisions de design [Bass et al., 2002]. Leur identification et l'analyse de leur importance dans le projet est une priorité pour l'architecte afin d'éviter de mauvaises décisions de design. Généralement, les attributs de qualité peuvent être identifiés en deux catégories : ceux impactant l'exécution (*e.g.*, performances, sécurité) qui sont exprimés dans les exigences des utilisateurs et ceux influençant la conception du projet (*e.g.*, maintenabilité) qui sont décrits dans les besoins des développeurs [Bertoa and Vallecillo, 2002].

Il est important de souligner que les différentes tâches d'analyse et de maintenance du backlog que nous venons de citer restent dans la plupart des cas des tâches manuelles coûtant énormément de temps [Bass, 2016] et leur bonne exécution est influencée par les limitations cognitives de ceux qui les réalisent.

## Solution

Notre objectif est de proposer un outil capable d'aider les architectes logiciel dans leur prise de décisions de design grâce à de l'extraction automatisée d'informations contenues dans les histoires d'utilisateur. Nous voulons faciliter la compréhension de la vue d'ensemble du projet le plus rapidement possible afin d'éviter certaines décisions de design erronées.

Nous pensons qu'une solution automatisée d'extraction et de modélisation d'informations pertinentes à la prise de décision de design permettrait un gain de temps et faciliterait l'analyse d'un backlog. Nous nous sommes tournés vers deux fonctionnalités, l'une aura pour but de détecter les histoires d'utilisateur faisant référence à des attributs de qualité et d'identifier ceux-ci et l'autre proposera une modélisation des histoires d'utilisateur composant les backlogs sous forme de diagrammes de robustesse.

Au cours des dernières années, l'apprentissage automatisé ou *machine learning* (ML) appliqué au développement logiciel a permis d'extraire des informations critiques à partir de documentation textuelle. Par exemple, nous pouvons citer les travaux de Bhat et son équipe [Bhat et al., 2017] qui ont permis d'extraire des décisions de design sur base des enjeux documentés d'un projet à l'aide du ML. Pour prouver que l'utilisation de modèles de classification ML est une solution envisageable pour notre première fonctionnalité, nous avons réalisé un Proof-of-Concept. Notre première intuition était qu'il nous faudrait utiliser des approches de classification avancée comme l'apprentissage profond. Pour vérifier cette intuition, nous l'avons comparée à d'autres modèles de classification (LinearSVC, ComplementNB, LogisticRegression) avec différents encodages (TF-IDF, Glove, BERT). Sur base de nos prédictions, nous avons proposé un classement par backlog des attributs de qualité les plus importants en fonction du nombre d'histoires d'utilisateur y faisant référence.

En développement logiciel, il est habituel d'avoir recours à des notations visuelles et des modélisations graphiques, *e.g.*, diagrammes conceptuels ou de domaine, pour faciliter la communication entre les différents intervenants [Verdi et al., 2002; Harel and Rumpe, 2004; Moody, 2009]. Nous pensons donc que l'utilisation de diagrammes permet de visualiser plus aisément un ensemble d'histoires d'utilisateur plutôt qu'une simple documentation textuelle. Dans notre cas, le diagramme de robustesse a été identifié comme un bon candidat car celui-ci permet de visualiser aisément une histoire d'utilisateur [El-Attar and Miller, 2010].

La deuxième partie de nos travaux consiste en l'extension du projet réalisé par Fabian Gilson et Calum Irwin [Gilson and Irwin, 2018]. Notre travail consistera à améliorer la version existante afin de la rendre utilisable sur un maximum d'histoires d'utilisateur. À l'aide du NLP, nous avons créé un parseur capable d'analyser une histoire d'utilisateur et d'en extraire les éléments importants afin de générer des diagrammes de robustesse.

Notre outil propose trois façon de modéliser les histoires d'utilisateur. La première génère le diagramme de robustesse d'une histoire d'utilisateur seule. La seconde unifie les diagrammes de robustesse de plusieurs histoires d'utilisateur afin de visualiser les relations entre les éléments qui les composent et par exemple de détecter toutes les histoires d'utilisateur ayant une relation. Pour faciliter la navigation à travers un diagramme de robustesse unifié trop grand, nous proposons l'utilisation de *"Points-De-Vue"* basées sur un élément du diagramme de robustesse. L'utilisateur pourra sélectionner un élement (Actor, Boundary ou Entity) du diagramme de robustesse unifié afin d'obtenir un diagramme de robustesse composé uniquement de éléments en relation avec ce dernier. Ces différents diagrammes de robustesse permettent aux architectes et à l'ensemble des intervenants une meilleure vue globale du projet à l'aide du diagramme principal et une meilleure analyse des liens entre les différents éléments du système à l'aide des diagrammes par *"Point-De-Vue"*.

## Publications

### Publication acceptée

Gilson, F., Galster, M., and Georis, F. (2019b). Extracting quality attributes from user stories for early architecture decision making. In **IEEE International Conference on Software Architecture Workshops)**, pages 129–136. IEEE

Galster, M., Gilson, F., and Georis, F. (2019). What quality attributes are present in product backlogs? a machine learning perspective. In **13th European Conference on Software Architecture (ECSA 2019)**

### Publication soumise

Gilson, F., Galster, M., and Georis, F. (2019a). Automatic generation and combination of visual use case scenarios from textual user stories. In **The 34th IEEE/ACM International Conference on Automated Software Engineering (ASE 2019)**

# Part I

# Etat de l'art

CHAPTER 1

# ANALYSE DU LANGAGE NATUREL ET APPRENTISSAGE AUTOMATISÉ




*Ce chapitre explique ce qu'est l'analyse du langage naturel, Natural Language Processing (NLP) et comment utiliser l'apprentissage automatique, Machine Learning (ML) dans celle-ci. Dans la première section, nous présenterons une vue d'ensemble des outils que nous offre le NLP. La deuxième exposera différents problèmes liés au langage naturel et comment utiliser le ML pour les résoudre. Dans la troisième, nous décrirons une brève histoire des différents types d'encodage. Dans la quatrième, nous présenterons un ensemble de techniques d'analyse de similarité. Nous terminerons par une comparaison des différentes librairies NLP en Python.*


## 1.1 Outils d'analyse du langage naturel

Lorsqu'un ordinateur est confronté à du texte, celui-ci n'a pas la capacité de faire de différence entre chaque mot et ne comprend pas non plus les dépendances qui existent entre eux. Dans cette section, nous allons présenter les différents outils nous permettant de résoudre cela. Nous les répartirons en deux catégories qui correspondent à deux étapes. D'abord, nous devons nettoyer et découper notre texte. Ensuite nous allons donner du sens aux mots composant notre texte.

### 1.1.1 Nettoyage et découpe du texte

Pour la première partie d'une pipeline NLP, nous allons présenter les différentes techniques dont nous disposons pour simplifier notre texte en retirant le bruit et en réduisant notre vocabulaire.

#### Retirer le bruit

Beaucoup de textes, principalement ceux issus de pages Internet, contiennent des symboles, chiffres et marqueurs qui ne correspondent pas à de l'information, mais à du bruit, *e.g.,* une balise HTML. Il est donc important de les retirer pour éviter une mauvaise tokenisation (défini au point suivant). Pour cela, il est nécessaire d'identifier les suites de caractères posant problème *e.g.,* l'ensemble des balises HTML et les supprimer.

#### Tokenisation

L'étape initiale d'un processus NLP est la tokenisation qui consiste en la découpe de phrases en tokens [Webster and Kit, 1992]. Un token correspond le plus souvent à un mot/symbole mais cela dépend de la méthode, des règles que l'on utilise et de nos objectifs. L'approche la plus naïve consiste à découper la phrase en fonction des espaces et des différents symboles de délimitation. Les processus plus intelligents utilisent une segmentation automatisée le plus généralement basée sur un dictionnaire et une stratégie de désambiguïsation [Borah et al., 2014].

*e.g., "As an archivist, I want to search images/profiles." =>*
*"As", "an", "archivist", ",", "I", "want", "to", "search", "images", "/", "profiles", "."*

#### Normalisation

Cette étape consiste à standardiser les mots qui composent la base de données pré-processée. Cela signifie que nous voulons utiliser une écriture commune pour chaque occurrence d'un mot, par exemple : "stopword", "stop word" et "stop-word". Malgré ces différentes formes, chacune elles fait référence au même mot. Donc, nous allons remplacer chaque occurrence par "stopword" qui sera notre standard. Cette étape est très utile pour les textes peu formels *e.g.,* des commentaires sur les réseaux sociaux.

#### Lemmatisation

Cette étape va consister en une simplification de nos tokens. Dans différentes langues, un même mot peut prendre plusieurs formes.

*e.g., "I am a player" et "We are some players"*

Nous pouvons constater que le verbe *"be"* va prendre la forme *"am"* et *"are"* en fonction de sa conjugaison. Le mot *"player"* va devenir *"players"* au pluriel. Pour un ordinateur, les deux formes de ce même mot sont identifiées comme différentes. La lemmatisation consiste à associer à chaque mot sa forme canonique pour permettre à la machine de comprendre que, par exemple, pour *"players"* malgré sa forme plurielle, il est fait référence au même mot ( *"player"*).

#### Identifier les mots d'arrêt et les supprimer

Les mots d'arrêt *ou StopWords* consistent en un ensemble de mots qui apparaissent très fréquemment. Ces mots sont souvent considérés comme du bruit et doivent donc être enlevés de la liste des tokens. Par exemple, en Anglais, nous pouvons retrouver les mots *"a", "the"*, etc. La méthode la plus simple et la plus fréquente consiste à utiliser une liste de mots d'arrêt faite à la main, un grand nombre de librairies offrant ce service *e.g.,* SpaCy[1], NLTK[2]. Dans certaines situations, les listes par défaut sont trop conséquentes ou pas assez, alors nous devrons en utiliser de celles qui ont été adaptées en ajoutant ou en retirant des mots. Par exemple, nous savons que certains mots spécifiques à notre domaine sont très récurrents et n'apportent pas d'information, donc nous pouvons les ajouter à cette liste.

### 1.1.2 Donner du sens à nos tokens

Pour la seconde partie d'un pipeline NLP, nous allons présenter différents outils permettant d'ajouter des propriétés à nos tokens comme leur type, leur rôle dans la phrase et les relations qu'ils entretiennent entre eux.

#### Prédiction du type de mot

Une première propriété intéressante pour un token est de connaître son type (Part-Of-Speech, POS) *e.g.,* nom propre, verbe et son TAG i.e. type spécifique d'un mot en fonction de son état (pluriel, conjugaison). Par exemple, pour un verbe, le TAG fera référence à sa conjugaison. Nous allons donc associer à chaque token un POS et un TAG. La méthode la plus efficace est d'utiliser un modèle ML entraîné à donner le POS/TAG d'un mot en fonction de celui-ci et de ceux qui l'entourent. Nous constatons que depuis 2005, l'état de l'art pour cette tâche obtient un score supérieur à 97% en Anglais [ACL, 2019]. Vu leur performance, il est possible de se fier à leur classification.

#### Détection des entités nommées

La détection des entités nommées, *Named Entity Recognition* (NER), est l'étape consistant à reconnaître les entités nommées et les associer à un type. Par exemple, "Marc lives in Brussels" : Marc est un nom de personne et Brussels est reconnu comme une ville. Les types les plus souvent labellisés sont les noms de personne, noms d'entreprise, lieux géographiques, noms de produit, dates, montants d'argent et les noms d'événement. Comme pour la prédiction des types de mot, nous utilisons un modèle de ML entraîné qui va labelliser nos données. Il est possible de customiser ce modèle en lui ajoutant des catégories *e.g.,* SpaCy est capable d'identifier les organisations, pays, etc. Pour cela, il lui faudra des données d'entraînement adaptées.

[1] https://spacy.io/
[2] https://www.nltk.org/

#### L'arbre des dépendances

Cette troisième propriété est très importante quand on a besoin d'analyser les liens entre chaque token. Pour réaliser cela, elle va se baser sur les étapes précédentes pour créer l'arbre des dépendances entre chaque mot. Elle va donc choisir un token racine qui possède des enfants tokens reliés à celui-ci par une relation *e.g.,* sujet d'un verbe, adjectif d'un mot,... Ces enfants auront eux aussi des enfants liés par une relation et ainsi de suite. Par le passé, nous avions recours à une liste de règles écrites à la main définissant leurs dépendances sur base du POS/TAG de chaque mot. Mais comme pour la prédiction du POS/TAG d'un mot, l'état de l'art nous montre qu'un modèle de ML est plus performant car beaucoup plus flexible. [Ruder, 2019]

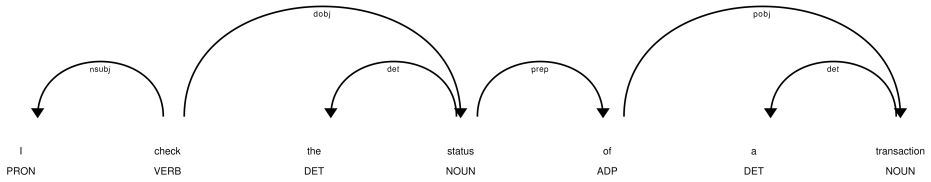


Figure 1.1: Arbre des dépendances créé par SpaCy

#### Co-référencement

Pour un être humain, l'association d'un pronom et du véritable mot qu'il remplace est relativement évidente mais faire comprendre cela à la machine est beaucoup plus difficile. A l'heure actuelle, le co-référencement reste l'un des challenges les plus compliqués en NLP. C'est la librairie java CoreNLP qui propose une des seules implémentations de l'état de l'art actuel avec un *F1-Score*[3] de 74% [Clark and Manning, 2016; NLP., 2019]. L'implémentation se rapprochant le plus de celle de CoreNLP en Python est celle proposée par l'équipe de HuggingFace qui se base sur les idées du papier de Stanford[Clark and Manning, 2016]. Pour proposer leur implémentation le plus facilement possible, ils se sont basés sur la librairie Spacy. [HuggingFace, 2019; Wolf, 2017]

## 1.2 L'apprentissage automatisé en NLP

### 1.2.1 Vue d'ensemble de l'apprentissage automatisé

L'apprentissage automatisé est devenu l'un des incontournables quand on parle d'intelligence artificielle. La principale vocation de ce domaine est de proposer diverses techniques pour permettre à la machine d'apprendre grâce aux données. On peut trouver 3 courants principaux [**?**]. Chacun d'eux se différencie par les données dont il aura besoin et la façon dont elles sont générées.

[3] *F1-Score* est une mesure de l'exactitude des prédictions

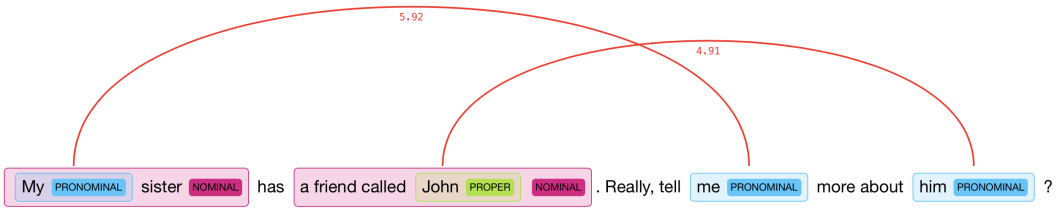


Figure 1.2: Exemple de Coref par la librairie de HuggingFace

Le premier, l'**apprentissage supervisé** a pour but de proposer des modèles capables de prédire un label/classe en fonction d'un ensemble de caractéristiques. Pour ce faire, les différents labels devront être connus et une base de données d'apprentissage labellisées devra exister.

Le second, l'**apprentissage non-supervisé** prédit des sous-ensembles dans une base de données en fonction des différentes caractéristiques de chaque donnée. Il ne nécéssite pas de données labellisées. Il est souvent utilisé pour explorer des données. Cela permettra d'identifier des classes ou labels encore ignorés ou confirmer des hypothèses.

Le dernier, l'**apprentissage par renforcement** a une approche très différente des deux premiers. Contrairement aux autres, il n'utilise pas de données d'apprentissage mais il va les créer lui- même en explorant un environnement. Il mémorise chaque décision qu'il a prise pour atteindre une cible et répétera l'exploration un grand nombre de fois pour trouver la meilleure solution à son problème. Un bel exemple existe dans le domaine des jeux de plateau où cette approche a permis de vaincre le champion du monde de GO grâce à l'IA AlphaGO. [Silver et al., 2018]

### 1.2.2 Classification en NLP

La première étape pour faire de la classification consiste à définir si elle est :

- **Simple** : chaque objet appartient à une et une seule classe.
- **Multi-Labelle** : un objet peut appartenir à 0 ou plusieurs classes.

Pour la classification simple, nous voulons un modèle capable de prédire un label parmi une liste définie au préalable. Il suffira donc d'entraîner un seul modèle sur un jeu de données labellisées et convertir ces labels en un vecteur composé de 0 sauf pour l'index associé au label.

La classification multi-label est plus complexe. Comme pour la version simple, il existe des modèles capables d'apprendre et de prédire une classification multi-labelle mais leur nombre est beaucoup plus restreint. Nous perdons des techniques efficaces comme le SVM [Chang and Lin, 2007; Hsu et al., 2003] qui a été pendant longtemps l'une des meilleures techniques de classification. Pour pouvoir continuer à utiliser l'ensemble de nos modèles, il existe la technique du *One-vs-the-rest* qui consiste à décomposer notre problème en une classification simple pour chaque label [Vapnik, 1995]. Il y aura donc un modèle pour chaque label entraîné à prédire si le label est présent ou non.

Ci-dessous, nous allons présenter une liste non exhaustive de différents modèles de classification supervisée. Pour chacun, nous en décrivons les grandes lignes suffisantes à leur compréhen-

sion générale.

#### Naive Bayes

Les techniques Naive Bayes [Zhang and Li, 2007] sont un ensemble d'algorithmes d'apprentissage supervisé basés sur l'application du théorème de Bayes [Wikipedia, 2019c] avec l'hypothèse "naïve" d'indépendance conditionnelle entre chaque paire de caractéristiques pour une classe. Un exemple d'implémentation est le ComplementNB[4]. Celui-ci est une adaptation de l'algorithme standard multinomial naïf de Bayes (MNB) [Rennie et al., 2003] particulièrement adapté aux ensembles de données déséquilibrées.

#### Régression Logistique

La Logistic Regression [Feng et al., 2014] est un modèle permettant la classification uniquement entre deux classes. Pour cela, il crée une fonction mathématique prenant en entrée les différentes caractéristiques d'un objet et prédira une valeur comprise entre 0 et 1. Chacune des deux bornes (0 et 1) correspond à une des deux classes. Il faudra définir une valeur de décision qui, si la valeur de la prédiction est supérieure, établira que l'objet fait partie de la classe 1 sinon de la classe 0. Lors de la phase d'entraînement, le modèle tentera de choisir la meilleure fonction en utilisant une fonction objectif qu'il optimisera grâce aux données d'entraînement. Cette optimisation aura pour but d'avoir le plus grand nombre de prédictions vraies.

#### Support Vector Machine

Le Support Vector Machine (SVM) [Chang and Lin, 2007; Hsu et al., 2003] a pour but de trouver un hyperplan dans un espace à N-dimension (N = le nombre de caractéristiques) qui séparera notre ensemble d'objets représentés dans cet espace en deux classes. Lors de sa phase d'apprentissage, le modèle va avoir comme objectif le fait de trouver le meilleur hyperplan parmi l'infinité de possibilités. Pour cela, il va choisir celle maximisant la marge, *i.e.* distance entre les deux plans parallèles à notre hyperplan passant respectivement par le point le plus proche de notre hyperplan faisant partie de la classe qu'il représente (Figure 1.3).

Ce modèle est efficace lorsque l'on a un grand nombre de dimensions et pour autant que ce dernier reste inférieur au nombre de données. L'une de ses faiblesses est qu'il ne peut pas résoudre les classifications multi-labels. Pour résoudre un problème de classification simple avec plusieurs classes, il est possible d'utiliser le Support Vector Classification (SVC) qui se base sur le SVM. Contrairement au SVM, il permet une découpe en plus de 2 classes.

#### Réseau de neurones

La technique du réseau de neurones consiste à utiliser un ensemble de couches de neurones interconnectées entre elles. Les neurones qui composent chaque couche sont l'application d'une fonction d'activation(ReLU, sigmoid, ...[Wikipedia, 2019a]) sur la somme des valeurs d'entrée

[4] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB

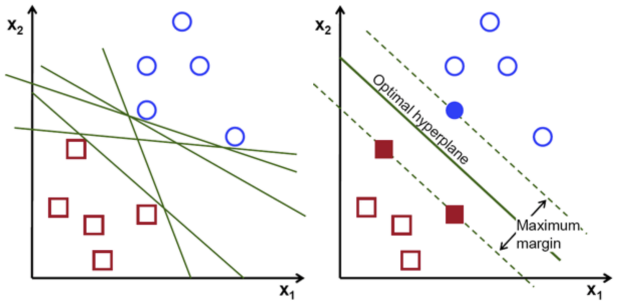


Figure 1.3: Ensemble des hyperplans possibles / choix de l'hyperplan optimal [Gandhi, 2018])

multipliées par leur poid respectif (w) (Figure 1.4). Chacun des neurones de chaque couche prend

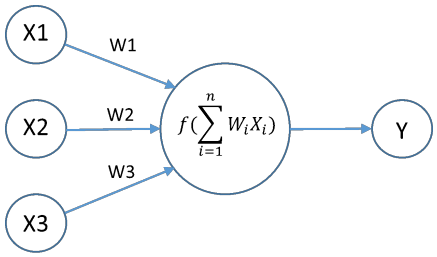


Figure 1.4: Calcul d'un neurone

comme valeur d'entrée l'ensemble des valeurs de sortie de la couche précédente. La dernière couche aura comme rôle de prédire la probabilité que l'objet en entrée appartienne à une classe. Pour cela, elle est composée d'un nombre de neurones équivalent au nombre de classes. La technique d'apprentissage de ce modèle s'appelle la backpropagation. Elle consiste à adapter les différents poids du réseau de neurones pour que les prédictions de celui-ci correspondent au mieux aux labels de la base de données d'entraînement.

#### Apprentissage profond

L'apprentissage profond a donné un second souffle aux réseaux de neurones et a permis de grandes avancées dans la résolution de différents problèmes NLP [Young et al., 2017]. Il consiste en la création de réseaux de neurones composés d'un grand nombre de couches. Ce type de modèle a vu naître un grand nombre d'architectures et différentes techniques. Les deux plus populaires sont :

- **Réseau convolutionnel** : est un réseau de neurones utilisant des couches convolutionnelles. Celles-ci sont composées de neurones qui ne sont pas connectés à l'ensemble des valeurs de sortie de la couche précédente. Pour le NLP, cette technique permet à chaque neurone de se concentrer sur des sous-ensembles de mots et créer des liens entre eux.

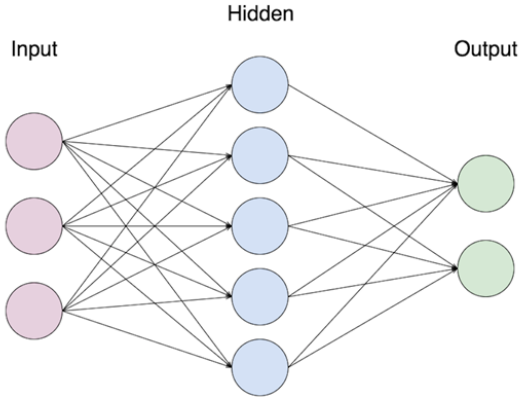


Figure 1.5: Réseau de neurones

- **Réseau récurrent** : dans les approches classiques, chaque valeur d'entrée est considérée comme indépendante. Les réseaux récurrents utilisent comme valeur d'entrée en plus de la valeur classique, la valeur de sortie pour l'objet précédent. Cela permet de créer un lien d'ordre et de dépendance entre les objets. On peut aisément comprendre son importance en NLP car les différents mots d'une phrase utilisés dans une autre peuvent avoir une signification différente.

Ces deux techniques sont considérées comme les plus efficaces [Yin et al., 2017]. On peut penser que les réseaux convolutionnels sont plus intéressants pour capturer les informations dépendant de caractéristiques statiques et les réseaux récurrents pour extraire une séquence de données. Mais les dernières études ont pu montrer que les dernières architectures se valent pour les différentes tâches. L'état de l'art change assez régulièrement entre ces deux approches [Yin et al., 2017].

### 1.2.3 Quelques métriques d'évaluation

Pour pouvoir comparer l'efficacité d'un modèle par rapport à un autre, il faut utiliser une métrique d'évaluation et choisir la meilleure en fonction de notre situation. Nous allons donc en décrire un ensemble ci-dessous.

Lorsqu'un modèle de classification effectue une prédiction sur un objet qui fait partie d'une classe ou non, sa prédiction est :

**Vrai Positif (TP)** : l'objet appartient à la classe prédite.
**Faux Positif (FP)** : l'objet n'appartient pas à la classe prédite.
**Vrai Négatif (TN)** : l'objet ne fait pas partie de la classe comme la prédiction le prévoit.
**Faux Négatif (FN)** : l'objet fait partie de la classe alors que la prédiction prévoyait le contraire.

Sur base de ces types de réponse, nous pouvons définir les métriques suivantes :

**Rappel**, taux de vrai positif : est le nombre d'objets correctement identifiés comme faisant

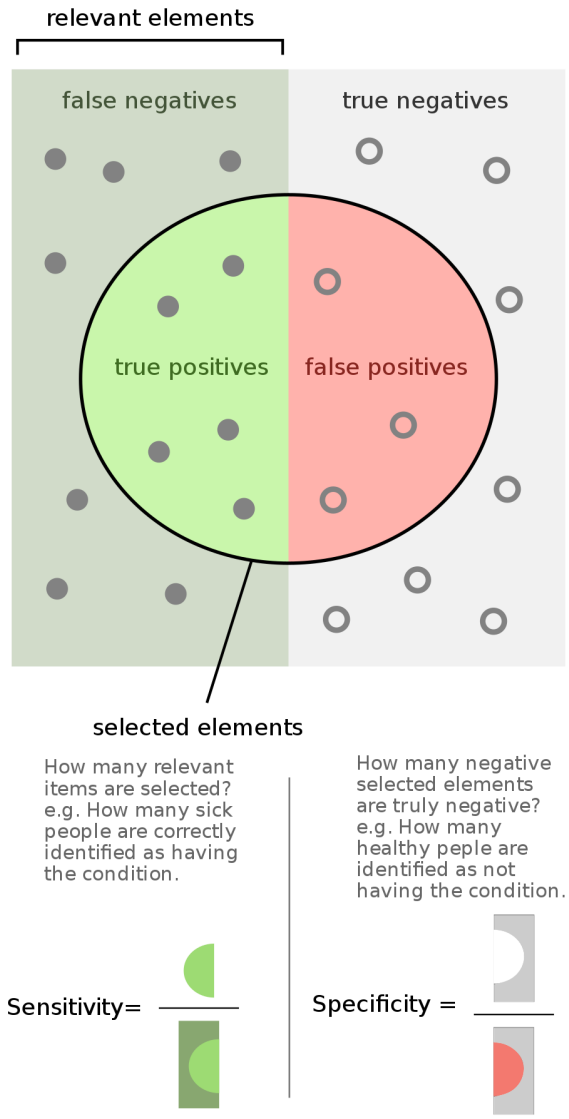


Figure 1.6: Répartition des prédictions entre TP, FP, FP, FN permettant de calculer la Spécificité et le Rappel [Wikipedia, 2019d])

partie la classe sur le nombre total d'objets dans celle-ci.

$$Rappel = \frac{VraiPositif}{VraiPositif + FauxNégatif}$$

**Spécificité**, taux de faux négatif, est le nombre d'objets correctement identifiés comme ne

faisant pas partie de la classe sur le nombre total d'objets dans celle-ci.

$$Spécificité = \frac{VraiNégatif}{VraiNégatif + FauxPositif}$$

***Précision*** : est le nombre d'objets correctement identifiés comme faisant partie de la classe sur le nombre total d'objets identifiés comme faisant partie de celle-ci.

$$Précision = \frac{VraiPositif}{VraiPositif + FauxPositif}$$

**Taux de faux positif**, Erreur de type 1 : est le nombre d'objets identifiés par erreur comme faisant partie de la classe sur le nombre total d'objets ne faisant pas partie de celle-ci.

$$TauxDeFauxPositifs = \frac{FauxPositif}{FauxPositif + VraiNégatif}$$

**Taux de faux négatif**, Erreur de type 2 : est le nombre d'objets identifiés par erreur comme ne faisant pas partie de la classe sur le nombre total d'objets faisant partie de celle-ci.

$$TauxDeFauxNégatifs = \frac{FauxNégatif}{FauxNégatif + VraiPositif}$$

**Efficacité** : est le pourcentage de prédictions correctes.

$$Efficacité = \frac{VraiPositif + VraiNégatif}{TP + TN + FP + FN}$$

***F1-Score*** : est une métrique très populaire pour évaluer la capacité d'un modèle à classifier. Elle doit sa popularité au fait qu'elle arrive à unir de façon efficace la *Précision* et le *Rappel*.

$$F1Score = 2 * \frac{Précision * Rappel}{Précision + Rappel}$$

Une manière efficace de valider nos résultats à l'aide des métriques ci-dessus est la ***k-fold cross-validation***. Celle-ci consiste d'abord à découper notre base de données en k parties (folds). Puis nous devons entraîner notre modèle avec k-1 parties et l'évaluer sur celles non utilisées. Nous devons reproduire cette opération pour les k parties de sorte que chacune ait été utilisée une et une seule fois pour évaluer le modèle. Finalement, il suffira de faire la moyenne des résultats obtenus lors des k évaluations pour obtenir les performances de notre modèle. Cette technique a pour but de réduire le biais causé par une évaluation unique sur une découpe unique de la base de données.

## 1.3 Vue d'ensemble des encodages

Lorsque l'on veut prédire une information en fonction de textes, comme les exemples précédemment cités de l'état de l'art le montrent, beaucoup de ces tâches sont réalisées par le biais de modèles d'apprentissage automatisé. Le problème de ces modèles, c'est qu'ils sont capables de lire uniquement des vecteurs de valeurs numériques. Il a donc fallu trouver des techniques pour convertir nos textes et nos mots en vecteur. Nous allons en présenter quelques unes des plus connues en commençant par les plus naïves jusqu'à l'état de l'art actuel.

### 1.3.1 Les ancêtres de l'encodage

#### Sac de mots

La technique la plus naïve, les sacs de mots ou *bag of words* (BoW) [Zhang et al., 2010], consiste à associer pour chaque mot un indice compris dans un vecteur de la taille du nombre de mots différents dans notre base de données. Pour représenter un mot, nous allons utiliser un vecteur composé de 0 sauf pour l'indice du mot qui sera à 1. Pour encoder une phrase, il suffira d'additionner tous les vecteurs représentant les mots qui la composent. Ce vecteur correspondra donc le nombre d'occurrences de chaque mot (Voir Figure 1.7).

| | Input | his | best | this | better | friend | a | fruit | player | for | the | is | buys | my |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | my friend is the best player | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| **1** | this player buys a better fruit for his friend | 2 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 0 | 2 | 1 | 0 |

Figure 1.7: Exemple d'encodage BoW

#### Term frequency–inverse document frequency (TF-IDF)

Le term frequency–inverse document frequency (TF-IDF) [Wu et al., 2008] est une statistique numérique destinée à refléter l'importance d'un mot ou d'un n-gramme, *i.e.* une sous-séquence de n caractères construite à partir d'un mot ou d'une phrase, pour un document d'une collection ou d'un corpus. La valeur augmentera proportionnellement à son nombre d'occurrences dans le document et sera diminuée par le nombre de documents contenant ce mot. Comme pour la méthode précédente, nous créerons un vecteur composé de l'ensemble des valeurs correspondantes pour chaque mot du corpus. Pour calculer la valeur d'un mot, il nous suffira donc de multiplier la fréquence du mot (TF) par la fréquence inverse par document (IDF).

$$TF(m) = \frac{Nombre\ d'apparition\ du\ mot\ m\ dans\ un\ document}{Nombre\ total\ de\ mots\ dans\ le\ document}$$

$$IDF(m) = log_e(\frac{Nombre\ total\ de\ documents}{Nombre\ total\ de\ documents\ contenant\ le\ mot\ m})$$

$$TF-IDF(m) = TF(m) * IDF(m)$$

### 1.3.2 L'encodage avec l'apprentissage profond

La majeure faiblesse de ces premières méthodes est que nos encodages ne prennent pas en compte le contexte du mot et que la taille des vecteurs générés peut devenir très grande en fonction de la taille de notre base de données. Nous allons donc présenter ci-dessous des techniques plus avancées permettant de résoudre certains de ces problèmes.

#### Word2Vec

Word2Vec est apparu pour solutionner les deux problèmes de ses prédécesseurs :

- Une taille fixée au début qui ne dépend plus du nombre de mots différents;
- Encodage créé en fonction du contexte et du mot.

Cette technique va utiliser un réseau de neurones pour encoder les mots, ce qui va nous permettre de définir la taille de l'output. Il existe deux principaux types d'architecture [Rong, 2014] :

- Modèle **Continuous Bag of Words (CBOW)** : cette première architecture va définir le vecteur du mot en fonction des mots composant son contexte. Il faut donc définir un nombre de mots représentant le contexte. Le modèle prendra donc en entrée les vecteurs représentant les mots du contexte et prédira un vecteur correspondant à celui du mot.
- Modèle **Skip-Gram** : est l'opposé de CBOW. Comme pour CBOW, nous définissons un contexte. Dans cette architecture, nous allons inverser les entrées et sorties. Nous allons donner en entrée le vecteur du mot actuel et en sortie les différents vecteurs des mots du contexte.

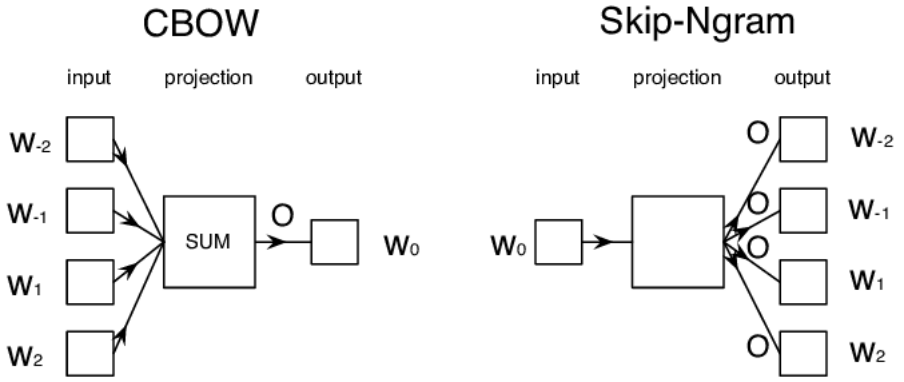


Figure 1.8: Architecture CBOW et Skip-Gram [Bornstein, 2018]

La façon la plus classique d'entraîner ces modèles est d'utiliser une énorme base de données de textes comme Wikipédia et de répéter l'opération sur toutes les données jusqu'à obtenir des performances optimales tout en évitant le surapprentissage ou *overfitting*i.e. les prédictions données par le modèle correspondent trop étroitement ou exactement aux données d'entraînement.

Après l'entraînement, un dictionnaire sera créé. Celui-ci sera composé de l'ensemble des mots utilisés pour l'entraînement et chacun sera associé au vecteur correspondant généré par le modèle.

Il a été montré que l'espace vectoriel dans lequel sont projetés les mots par Word2Vec réussit à capturer de façon efficace la syntaxe et la sémantique des mots [Mikolov et al., 2013]. De cela découle une proximité vectorielle entre 2 mots similaires ou souvent utilisés dans les mêmes cas et la représentation de certaines relations *e.g.*, masculin-féminin, conjugaison d'un verbe, capitale-pays.

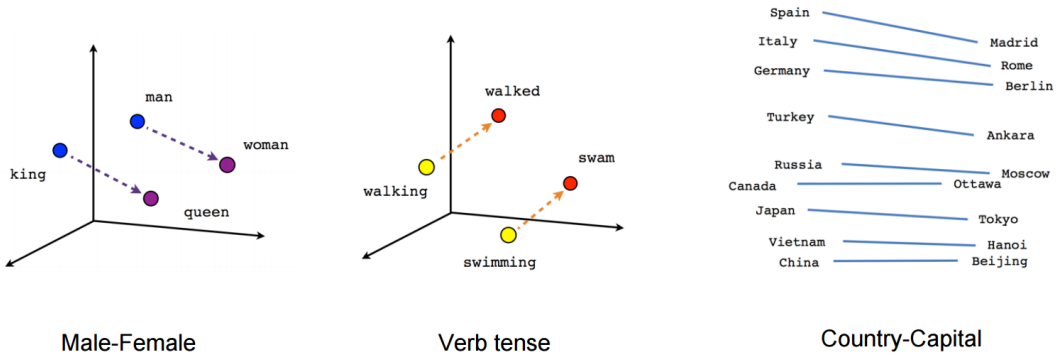


Figure 1.9: Illustration des différentes relations partagées entre les vecteurs générés par Word2Vec [Tensorflow, 2019])

#### Global Vectors for Word Representation

Global Vectors for Word Representation (Glove) [Pennington et al., 2014] veut apporter une approche plus globale que Word2Vec. Word2Vec va s'entraîner sur un flux de phrases, ce qui l'empêchera de prendre en compte les informations contenues dans l'ensemble du texte. Pour résoudre cette faiblesse, Glove va se baser sur une matrice de co-occurrence globale ($X$). Celle-ci correspond au nombre d'occurrences d'un mot apparaissant dans le contexte d'un autre mot sur l'ensemble de notre base de données($X_{ij}$). Le contexte d'un mot est la liste des mots voisins à celui-ci, comme pour Word2Vec.

Il suffira de minimiser sa fonction objectif pour créer nos vecteurs. Ils ont défini une containte simple :

$$w_i^T w_j + b_i + b_j = log(X_{ij})$$

Ici $w_i$ - vecteur pour le mot principal, $w_j$ - vecteur pour le mot de contexte, $b_i$, $b_j$ sont des biais scalaires pour les mots principaux et de contexte.

Voici la fonction objectif de Glove :

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

$f$ est la fonction de pondération qui nous aide à éviter d'apprendre uniquement à partir de paires de mots extrêmement communes. Les auteurs de GloVe ont choisi la fonction suivante :

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^{\alpha} & \text{if } X_{ij} < XMAX \\ 1 & \text{otherwise} \end{cases}$$

Ici, $\alpha$ est un métaparamètre ajustable.

En minimisant notre fonction objectif, les vecteurs captureront les différentes propriétés sémantiques reprises dans la probabilité qu'un mot soit présent dans le contexte d'un autre.

Comme pour Word2Vec, Glove créera un dictionnaire avec un encodage pour chaque mot. Selon les créateurs de Glove, ses résultats sont très similaires à ceux de Word2Vec [Pennington et al., 2014]. Il capture lui aussi les différentes propriétés sémantiques citées pour Word2Vec. Sa force est qu'il est plus rapide à entraîner, ce qui favorise son utilisation dans l'implémentation d'une pipeline NLP [Pennington et al., 2014].

#### FastText

L'ensemble des techniques que nous avons présentées précédemment considèrent chaque mot comme différent et lui associent un vecteur propre. Cette façon de raisonner est limitée car en Anglais ou en Français, beaucoup de mots partagent une partie commune *e.g.*, homme et bonhomme. Une autre limite des approches précédentes est que pour avoir l'encodage d'un mot, il faut qu'il soit au moins apparu une fois dans la base de données d'entraînement. FastText [Bojanowski et al., 2016] se base sur l'approche de Skip-Gram présentée précédemment, la différence majeure étant au niveau de la représentation d'un mot.

Dans cette technique, on n'associe pas un vecteur à chaque mot mais à chaque n-gramme (liste de n caractères) trouvé dans la base de données d'entraînement. Pour chaque mot, nous aurons un ensemble de n-grammes qui le représente.

*"where"* => <*wh , whe , her, ere, re*>

Le vecteur de chaque mot sera la somme des vecteurs de ses n-grammes. La force de cet encodage est qu'il obtient de meilleurs résultats pour les mots peu fréquents et une meilleure adaptation au nombre de mots différents dans les données d'entraînement [Bojanowski et al., 2016].

### 1.3.3 Les dernières architectures d'apprentissage profond

Les modèles présentés dans la partie précédente définissent un encodage pour chaque mot en fonction d'une base de données. Le problème de ce mapping est qu'on a un même encodage pour un mot qui possède plusieurs significations et qu'il n'y a pas de prise en compte du contexte actuel qui pourrait influencer la signification actuelle du mot. Ci-dessous, nous allons présenter les dernières avancées de l'état de l'art pour l'encodage des mots.

#### ELMo

ELMo [Peters et al., 2018a] est une technique de représentation du langage qui capture les propriétés complexes de chaque mot, comme la syntaxe et la sémantique, et les adapte en fonction du contexte dans lequel est utilisé le mot. Ceci permet d'éviter un encodage similaire pour un même mot utilisé dans 2 contextes totalement différents. Pour faire cela, il se base sur un modèle bidirectionnel d'apprentissage profond (biLM) qui est entraîné sur une grande base de données de textes, *e.g.,* l'ensemble des articles de Wikipédia. Lors de sa création, cet encodage a été testé à l'aide de modèles permettant de réaliser différentes tâches NLP, *e.g.,* identification du POS, NER, et il a pu montrer des améliorations significatives par rapport à l'état de l'art [Peters et al., 2018a].

#### Bidirectional Encoder Representations from Transformers

L'état de l'art actuel, Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2018] est une technique de représentation du langage qui, comme ELMo, capture les propriétés complexes de chaque mot, comme la syntaxe et la sémantique, et les adapte en fonction du contexte dans lequel est utilisé le mot. Il a été créé pour prendre en compte le contexte actuel de chaque côté du mot, contrairement à d'autres approches unidirectionnelles [Peters et al., 2018b]. Pour capturer les relations contextuelles entre différents mots, ses créateurs ont utilisé un Transformer. Il est composé d'un réseau de neurones encodeur et d'un autre décodeur. L'encodeur entraîné permettra d'extraire le contexte d'un nouveau mot. Pour prouver son efficacité, il a été évalué sur les différents défis du NLP, *e.g.,* NER, et a démontré une amélioration significative pour la plupart d'entre eux [Devlin et al., 2018].

## 1.4 Vue d'ensemble des techniques d'analyse de similarité

Ci-dessous, nous allons présenter une liste non-exhaustive de différentes métriques/techniques permettant de mesurer la similarité entre deux mots.

**Levenshtein Algorithm** : calcule la distance entre deux mots comme le nombre minimum de caractère seul qu'il est nécessaire d'éditer (insérer, supprimer ou modifier) pour, à partir d'un mot, obtenir le deuxième.

**Comparaison par n-gramme** [Kondrak, 2005] : cette méthode consiste à diviser un mot en sous-ensembles de N caractères (n-gramme) et de calculer la similarité comme le nombre de tri-gramme unique partagé entre les deux mots divisés par le nombre de tri-gramme unique qui les composent. Par exemple pour N = 3 :

$$\textit{"woman"} = \textit{"wom" oma" man"}$$
$$\textit{"womna"} = \textit{"wom" "omn" mna"}$$

wom est le seul trigram partagé donc la similarité = 1 / 5. Le problème dans cette découpe est que l'on ne prend pas en compte le fait que les deux mots partagent ou non leurs premiers et derniers caractères. Pour résoudre ce problème, on ajoute un caractère spécial pour créer un trigram de début et de fin du mot différent de ceux composant l'intérieur du mot.

*"woman" = "<wo" "wom" "man" "an>"*

**Jaro Algorithm** [Store, 2018] : calcule la distance entre deux mots comme le nombre minimum de transpositions de simple caractère nécessaire pour passer d'un mot à l'autre.

$$d_j = \frac{1}{3}(\frac{m}{|\,s_1\,|} + \frac{m}{|\,s_2\,|} + \frac{m-t}{m})$$

où

- $|\,s_i\,|$ est la longueur de la chaîne de caractères $s_i$;
- $m$ est le nombre de caractères correspondants entre les 2 mots;
- $t$ est le nombre de transpositions pour que les 2 mots soient identiques.

Par exemple, $s_1$ MARTHA et $s_2$ MARHTA, on trouve [5]:

- $|\,s_1\,| = 6$
- $|\,s_2\,| = 6$
- $m = 6$
- $t = \frac{2}{2} = 1$

On obtient comme distance de Jaro :

$$d_j = \frac{1}{3}(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6}) = 0,944$$

**Jaro-Winkler Algorithm** se base sur l'approche de Jaro en ajoutant la prise en compte du préfixe partagé entre les 2 mots.

$$distance\ JaroWinkler = d_j + lp(1-d_j)$$

On obtient comme distance de Jaro Winkler pour l'exemple précédent avec un $p = 0,1$ et un $l = 3$ :

$$d_{JW} = 0,944 + 3 * 0,1(1-0,944) = 0,961$$

**Liste de synonymes** : on mesure la similarité de façon binaire. La similarité entre 2 mots est égale à 1 si le mot fait partie de la liste des synonymes de l'autre mot, sinon 0. Il est possible d'utiliser des listes de synonymes pré-existantes proposées par certaines librairies *e.g.,* NLTK.

**Proximité vectorielle** : l'une des propriétés capturées par les encodages comme Word2Vec, Glove, etc. est que deux mots employés dans des contextes similaires auront des vecteurs proches. Pour calculer la similarité entre deux mots, il suffit de calculer la proximité vectorielle entre leurs 2 vecteurs pour savoir à quel point les 2 mots sont similaires.[Mikolov et al., 2013] Par exemple, SpaCy[6] utilise la similarité par Cosine [Wikipedia, 2019b].

$$\cos(t,e) = \frac{te}{\|t\|\|e\|} = \frac{\sum_{i=1}^{n} t_i e_i}{\sqrt{\sum_{i=1}^{n}(t_i)^2}\sqrt{\sum_{i=1}^{n}(e_i)^2}} \tag{1.1}$$

où $t$ et $e$ sont des vecteurs représentant chacun un mot.

[5] https://fr.wikipedia.org/wiki/Distance_de_Jaro-Winkler
[6] https://spacy.io/api/token

## 1.5 Vue d'ensemble des librairies Python NLP

Dans cette section, nous allons présenter une liste non-exhaustive des librairies les plus populaires pour résoudre des tâches NLP en Python. Nous ne ferons pas allusion à Tensorflow[7], Keras[8] et Pytorch[9] qui sont les librairies les plus connues pour réaliser de l'apprentissage profond car celles-ci sont difficilement maîtrisables et ne sont pas centrées sur le NLP.

### 1.5.1 Natural Language ToolKit

NLTK[10] est l'une des librairies les plus connues en NLP et l'une des plus anciennes, restée très populaire. Elle propose une grande partie des outils NLP cités dans la Section 1.1 *e.g.*, tokenization, lemmatization, ... et une grande variété d'implémentation de classifieurs. Elle est capable de gérer un très grand nombre de langues. Ses principales faiblesses sont sa difficulté à être apprise et sa lenteur sur certaines tâches [Bobriakov, 2018]. Malheureusement, malgré son grand panel d'outils, les réseaux de neurones et Word2Vec (ou autres techniques utilisant de l'apprentissage profond) ne sont pas intégrés.

### 1.5.2 SpaCy

SpaCy[11] est devenue assez rapidement une des incontournables lorsque l'on parle de NLP. Sa force réside dans sa simplicité et dans sa rapidité [Al Omran and Treude, 2017]. Elle propose une pipeline avec des modèles pré-entraînés dans 7 langues. Pour utiliser ses services, il suffit d'importer le modèle correspondant à la langue du texte et d'y insérer le texte. SpaCy s'occupe elle-même de créer les tokens, l'arbre des dépendances et le POS. SpaCy se base sur des réseaux de neurones profonds convolutionnels pour ces différentes tâches. Elle permet d'entraîner nos propres modèles de classification ou de NER très simplement. Elle utilise Glove par défaut pour encoder les mots. Contrairement à d'autres librairies, SpaCy utilise le plein potentiel de l'orienté objet en convertissant notre texte en différents objets faciles à manipuler (Token, Doc , ...).

### 1.5.3 Scikit-learn

Scikit-learn[12] est l'une des librairies les plus populaires pour débuter le ML en général. Elle propose une multitude d'outils utiles comme des métriques, des outils de découpe de base de données, ... Spécifiquement pour le NLP, elle nous propose les techniques d'encodage classique pour le sac de mots ou TF-IDF. Scikit-learn est une librairie généraliste pour le ML. Elle met à disposition une multitude de modèles de classification mais ne propose pas de modèles d'apprentissage profond. Elle se limite aux réseaux de neurones simples. Etant une librairie créée pour les débutants, elle

[7] https://www.tensorflow.org
[8] https://keras.io
[9] https://pytorch.org
[10] https://www.nltk.org
[11] https://spacy.io
[12] https://scikit-learn.org/stable/

dispose d'une documentation très fournie et beaucoup d'exemples d'utilisation facilitant sa prise en main.

### 1.5.4 Gensim

Gensim[13] est une librairie supportant l'apprentissage profond. Elle est fournie avec différentes techniques d'encodage *e.g.,* TF-IDF, Word2Vec, document2Vec, etc. Sa faiblesse réside dans le fait qu'elle ne fournit pas les différents outils NLP utiles comme la tokenisation. Donc, elle ne se suffit pas à elle-même pour créer une pipeline complète comme NLTK ou Spacy le font.

### 1.5.5 Pattern

Pattern[14] est une librairie créée pour parser des pages Web. Elle propose un ensemble d'outils et de classifieurs comme la prédiction des POS, recherche par n-gramme pour l'analyse de sentiments, WordNet, ... Elle possède des connections pré-créées avec certaines API *e.g.,* Facebook, Twitter, etc. Son optique de parseur du Web la rend plus difficile à utiliser et moins optimale pour des tâches de NLP pur.

### 1.5.6 Flair

Flair[15] est une librairie assez récente proposant un framework permettant d'utiliser simplement des implémentations de l'état de l'art en NLP *e.g.,* BERT, ELMo etc. Selon ses créateurs, sa force réside donc dans sa simplicité d'utilisation et du fait qu'elle propose les dernières avancées de l'état de l'art. Elle permet comme Spacy d'utiliser un panel d'outils NLP (POS, NER, ...). Son nombre de langues supportées reste limité mais s'élargit néanmoins assez rapidement grâce à sa grande communauté. Elle est basée sur la librairie Pytorch, ce qui permet de l'intégrer facilement à d'autres outils proposés par cette librairie bien connue pour l'apprentissage profond.

### 1.5.7 Comparaison

Selon Al Omran et C. Treude [Al Omran and Treude, 2017], Spacy est un des meilleurs choix du fait de sa prise en main très rapide et son efficacité sur les différentes tâches NLP. Elle n'est toutefois pas la meilleure car elle n'intègre pas les dernières avancées de l'état de l'art. Une première expérience positive dans le cadre d'un autre projet nous a confortés dans ce choix pour prouver rapidement la faisabilité de notre idée.

## 1.6 Conclusion

L'apprentissage automatisé en NLP propose une très grande variété d'outils et d'alternatives permettant d'accomplir diverses tâches sur nos textes *e.g.,* nettoyage, arbre des dépendances,

[13] https://radimrehurek.com/gensim/
[14] https://github.com/clips/pattern
[15] https://github.com/zalandoresearch/flair

encodage, classification, similarité, etc. Actuellement, l'un de nos meilleurs alliés permettant d'utiliser simplement et rapidement une grande partie de ces techniques est la librairie SpaCy. Malheureusement, celle-ci ne propose qu'un type d'encodage (Glove) et un type de modèle de classification (réseaux de neurones convolutionnels). Des alternatives comme Scikit-learn nous proposent un large panel de classifieurs du ML et des encodages comme TF-IDF. De plus, la librairie Flair, nous permettra d'expérimenter les dernières avancées dans l'état de l'art (BERT). Dans notre contribution, nous nous focaliserons sur SpaCy pour le préprocessing (tokenisation, POS, arbre des dépendances, ...) et nous comparerons les performances de son classifieur à celles d'autres techniques du ML en utilisant différents encodages (TF-IDF, Glove et BERT) pour nos données.

CHAPTER 2

# DÉVELOPPEMENT LOGICIEL AGILE

*Dans ce chapitre, nous allons présenter la méthode de développement logiciel Agile, les histoires d'utilisateur et le développement d'une architecture logiciel. Dans la première section, nous allons expliquer les origines de la méthode Agile et en quoi elle consiste. Dans la deuxième, nous expliquerons ce qu'est une histoire d'utilisateur et différents Frameworks à utiliser avec celle-ci. Ensuite, nous présenterons ce qu'est une décision de design et nous nous attarderons brièvement sur ce qui les influence. Nous terminerons par décrire comment visualiser des exigences.*

## 2.1 Agile

### 2.1.1 La naissance d'Agile

Dans les débuts du développement logiciel, les exigences des clients étaient rapidement fixées dès les premières étapes du projet [Malik Hneif, 2009]. Les techniques de développement pour rèpondre aux besoins du client étaient séquentielles comme le "Watterfall". Celles-ci consistaient à découper le développement en une suite de phases, chacune devant être validée pour entamer la suivante (définir les exigences, choisir l'architecture, implémenter, tester et maintenir). Cette approche était donc très peu modulaire ; elle nécessite la rédaction d'un cahier des charges clair au début du projet qui ne pourra plus être modifié. Ce manque de flexibilité est devenu de plus en plus contraignant vu les nouvelles exigences clients. L'évolution des exigences clients a causé l'apparition de nouvelles difficultés auxquelles les approches séquentielles peu flexibles avaient du mal à répondre [Malik Hneif, 2009] :

- les exigences/besoins des clients changent avec le temps en fonction du contexte économique, des lois, de l'évolution de leurs activités ,...

- mauvaise compréhension des exigences lors de la rédaction du cahier des charges, concept trop vague, manque de connaissances du domaine,...
- la mise sur le marché d'un logiciel doit être de plus en plus rapide tout en étant le moins coûteux possible vu que le marché est de plus en plus compétitif;
- manque d'implications des clients dans le développement du projet causé par le manque d'interaction avec l'équipe de développement après la rédaction du cahier des charges.

Pour mieux répondre à ces nouvelles contraintes, un ensemble d'approches plus itératives ont commencé à voir le jour. La première référence à ce type de processus a été faite dans un rapport de Zurcher et Randell en 1968 [Zurcher and Randell, 1968]. Ils ont proposé une méthode dans laquelle le designer affinait la définition d'un composant du système à travers plusieurs itérations. En parallèle, Royce a défini l'approche Watterfall qui consiste en une suite d'étapes séquentielles à exécuter deux fois. La première itération releve de la création d'un prototype à présenter au client et la seconde au développement du produit final [Royce, 1970]. Plus le temps passait, plus le nombre de nouvelles approches de plus en plus flexibles voyaient le jour. Suite à la naissance de ces idées, une partie de communauté des développeurs s'est rassemblée afin de définir une nouvelle méthode de développement logiciel répondant le mieux possible aux nouveaux enjeux. Pour cela, ils ont rassemblé les points positifs de chacune de ces nouvelles idées afin de créer la technique de développement logiciel Agile. [Malik Hneif, 2009; Kent Beck, 2001]

### 2.1.2 Le manifeste d'Agile

L'alliance Agile a voulu définir la méthode Agile à l'aide d'un manifeste. Celui-ci nous montre que Agile repose sur 4 concepts clefs [Kent Beck, 2001] :

- **"Les individus et leurs interactions** plus que les processus et les outils"
- **"Des logiciels opérationnels** plus qu'une documentation exhaustive"
- **"La collaboration avec les clients** plus que la négociation contractuelle"
- **"L'adaptation au changement** plus que le suivi d'un plan"

Il découle, à partir de ces 4 concepts clefs, les 12 principes agiles :

- "Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée."
- "Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client."
- "Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts."
- "Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet."
- "Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés."
- "La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face."
- "Un logiciel opérationnel est la principale mesure d'avancement."

- "Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant."
- "Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité."
- "La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle."
- "Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées."
- "À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence."

### 2.1.3 Agile en pratique

Dans Agile, nous avons deux groupes d'acteurs qui devront interagir entre eux.

**Les clients/intervenants** : c'est l'ensemble des personnes qui ont leur mot à dire sur le projet et/ou qui vont interagir avec celui-ci. On peut retrouver dans cette catégorie les utilisateurs finaux du logiciel, les personnes demandant le développement du projet, les acteurs légaux qui pourraient avoir une influence.

**Les développeurs** : ce sont toutes les personnes qui composent l'équipe de développement. Ce sont elles qui devront répondre aux exigences du client par le développement des différentes fonctionnalités nécessaires.

En Agile, nous conservons l'ensemble des exigences du projet dans un backlog, liste composée des exigences. Contrairement à la méthode "Waterfall", en Agile nous voulons un développement continu du produit et des feed-backs réguliers de la part des clients. Il existe une multitude de pratiques pour accomplir cela. Prenons par exemple celle proposée par le Framework Scrum qui consiste en un processus incrémentale divisé en cycles de courte durée. Chaque itération est composée de ces étapes [Schwaber and Beedle, 2001]:

(1) Rencontrer le client pour définir les ajouts/modifications/suppressions dans le backlog.
(2) Planifier les US à satisfaire à la fin du cycle en fonction de leur valeur business et de leurs difficultés à être implémentées et identification/résolution des risques liés à celles-ci.
(3) Choisir les designs utilisés pour intégrer les nouvelles fonctionnalités remplissant les US tout en tenant compte des risques liés.
(4) Développer/Tester/Déployer en continu les fonctionnalités en respectant les designs choisis.
(5) Faire évaluer les nouvelles fonctionnalités par les clients et prendre en compte leurs remarques dans le backlog.

### 2.1.4 Histoire d'utilisateur

Une histoire d'utilisateur, user storie (US), est la description textuelle d'une fonctionnalité du système en langage naturel *e.g.*, le français ou l'anglais. Elle doit être écrite par les développeurs ou par le client dans un langage compréhensible par le client car chacune d'elle doit être validée par celui-ci, donc le jargon technique est à proscrire. Elle respecte le plus souvent le template [Cohn, 2004] :

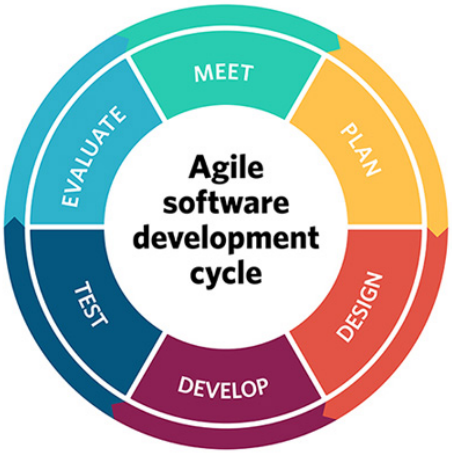


Figure 2.1: Cycle de vie Scrum [Schwaber and Beedle, 2001])

*"**As a** <acteur>, **I want** <objectif> ,[**so that** <bénéfice>]"*

Cette technique de représentation des exigences est devenue une des plus populaires grâce à son efficacité et elle est le plus souvent utilisée dans le développement logiciel Agile. [Wautelet et al., 2014b; Lucassen et al., 2016b] La liste des US d'un projet sera contenue dans le backlog de ce projet. Le client ainsi que les développeurs associeront à chaque US une valeur business et parfois une estimation de la durée nécessaire au développement de celle-ci. [Cohn, 2004]

### 2.1.5 Quality User Story Framework

Malgré cette popularité, on peut constater que la qualité d'écriture des US est souvent médiocre et entraîne divers problèmes [Schön et al., 2017; Inayat et al., 2015]. C'est pour cela que G. Lucassen et son équipe proposent un framework composé de 14 critères de qualité permettant d'augmenter la qualité de chacune d'elles [Lucassen et al., 2015]. Ces critères se divisent en trois types : Syntaxique, Sémantique et Pragmatique.

Les qualités syntaxiques font référence aux problèmes de structures textuelles dans la US sans prendre en compte le sens.

- "**Atomique** : une US exprime une exigence pour exactement une fonctionnalité."
- "**Minimale** une US ne contient rien de plus que le rôle, les moyens et les fins."
- "**Bien formée** : une US comprend au moins un rôle et un moyen."

Les qualités sémantiques : font référence au sens et aux relations entre les différentes parties de l'US.

- "**Sans conflit** : une US ne doit pas être en contradiction avec toute autre histoire d'utilisateur."
- "**Conceptuellement solide** : La partie centrale exprime une caractéristique et la fin exprime une raison, pas autre chose."
- "**Orienté problème** : une US ne spécifie que le problème, pas la solution."

- "**Sans ambiguïté** : une US évite les termes ou les abstractions qui peuvent donner lieu à de multiples interprétations."

Les qualités pragmatiques : font référence au meilleur choix de mots pour communiquer l'ensemble des exigences.

- "**Complète** : une US se suffit à elle-même pour implémenter une fonctionnalité complète. Elle ne dépend d'aucune autre US pour être implémentée."
- "**Dépendances explicites** : les différentes dépendances dans une US ne doivent pas faire place à l'hésitation, elles doivent être claires."
- "**Phrase complète** : une US est une phrase complète bien formée (respectant les règles d'écriture de la langue dans laquelle elle est écrite)."
- "**Indépendante** : une US est autonome, évitant les dépendances inhérentes à d'autres histoires d'utilisateur."
- "**Estimable** : une US ne décrit pas des exigences trop grossières/vagues qui sont difficiles à planifier et à classer par ordre de priorité."
- "**Uniforme** : toutes les US suivent le même template d'écriture."
- "**Unique** : toutes les US sont uniques, les doublons sont à éviter. "

Sur base de leur Framework défini ci-dessus, Lucassen et son équipe ont développé AQUSA [Lucassen et al., 2016a]. AQUSA est un outil capable de détecter les critères du framework non-respectés et de proposer des alternatives pour les résoudre.

### 2.1.6 Framework INVEST

INVEST est un framework permettant d'améliorer l'efficacité de rédaction des US. Il apporte une liste de principes à respecter pour avoir une US la plus précise et compréhensible possible. Les voici [Wake, 2003]:

- **I- Independent** une US ne doit pas dépendre d'une autre US pour être compréhensible, elle doit se suffire à elle-même et doit pouvoir être implémentée indépendamment des autres.
- **N - Negociable** une US doit pouvoir être négociable, elle ne doit pas être trop détaillée, elle doit seulement exprimer l'exigence. Les détails seront définis entre le client et les développeurs pendant le processus de développement.
- **V- Valuable** une US doit dégager une valeur pour le client. Il doit être capable de comprendre son importance pour lui donner une valeur.
- **E- Estimable** une US doit être assez claire pour qu'il soit facile d'estimer la complexité et la diviser en tâches.
- **S- Small** une US doit être courte pour faciliter sa clarté et son estimation.
- **T- Testable** une US doit être testable. On doit donc savoir clairement définir comment interagir avec elle. Pour cela, elle peut être complétée par des critères d'acceptations qui décrivent les inputs et outputs attendus.

## 2.2 Architecture logiciel

À travers l'histoire du développement logiciel, un grand nombre de définitions pour l'architecture logiciel (AL) ont été proposées. Fabian Gilson nous suggère une définition créée sur base de l'unification des définitions données par Perry and Wolf and by Gacek et al. supersedes [Gilson, 2015; Perry and Wolf, 1992; Gacek et al., 1995].

Nous pouvons donc représenter une AL comme les définitions **structurelles et comportementales** des constructions architecturales, les **relations** entre ces éléments ainsi que les **structures alternatives**, la **rationalité du design**, l'intégration des choix de conception et les motivations qui soutiennent l'architecture résultante et les **liens** avec les **exigences** des intervenants.

### 2.2.1 Décision de design

Nous allons plus particulièrement nous concentrer sur les décisions de design qui sont l'un des points les plus importants qui composent l'AL. L'importance est telle que certains chercheurs résument l'AL comme un ensemble des décisions de design prises pour l'implémentation, *e.g.*, choix des patterns architecturaux, stratégie ... [Jansen and Bosch, 2005; Vliet and Tang, 2016; Jansen, 2008]

Une décision de design relève d'un choix rationnel débattu dans l'équipe de développement pour sélectionner la meilleure option d'implémentation à un moment donné dans le processus de développement [Vliet and Tang, 2016].

Malheureusement, dans la réalité, un choix est rarement rationnel car le manque de temps et d'informations influencera notre décision et l'empêchera d'être totalement rationnelle [Tang et al., 2017].

Une décision de design est donc un choix "rationnel limité" comme Herbert Simon l'a défini [Simon, 1996] : "Dans la prise de décision, la rationalité des individus est limitée par l'information dont ils disposent, les limites cognitives de leur esprit et le temps limité dont ils disposent pour prendre une décision.".

Pour réduire ce problème de rationalité limitée, certains chercheurs ont voulu trouver des moyens d'automatiser la prise de décision de design ou d'extraire de façon automatique les informations les plus importantes pour ces choix [Bhat et al., 2017; Shahbazian et al., 2018; Lu and Liang, 2017].

#### Attributs de qualité

Les **attributs de qualité** sont les exigences non-fonctionnelles qu'un logiciel doit remplir telles que la performance, la sécurité, la compatibilité, l'utilisabilité, la fiabilité, la maintenabilité ou la portabilité [ISO/IEC, 2010]. Selon une grande partie des architectes logiciels, ceux-ci ont une importance indéniable dans les décisions de design architectural. Il est donc essentiel de les identifier le plus rapidement possible pour éviter de mauvais choix architecturaux [Bass et al., 2002]. Les QAs peuvent être séparés en deux types :

- **Les attributs de qualité à l'exécution** (*e.g.*, performances, security) sont les QAs qui influencent l'expérience utilisateur lors de l'exécution du système. Ceux-ci sont généralement exprimés dans les exigences des utilisateurs..
- **Les attributs de qualité à la conception** (*e.g.*, maintainability) englobent les QAs qui influencent la manière de concevoir le système par les développeurs. Ceux-ci sont généralement décrits dans les besoins des développeurs pour maintenir le logiciel au cours de son cycle de vie.

## 2.3 Visualisation des exigences

Comme nous l'avons présenté précédemment, en Agile, l'implication des utilisateurs et des intervenants extérieurs est primordiale. C'est pour cela que l'on utilise les US pour décrire les exigences d'un projet en langage compréhensible par les deux parties. Plus leur nombre est important, plus il devient compliqué pour les acteurs de rester impliqués. Pour simplifier la compréhension du backlog, les développeurs ont souvent recours à diverses techniques de modélisation (Diagramme de classe, etc). Malheureusement, celles-ci restent trop souvent très dificiles à comprendre pour les non-initiés à cause de leur notation peu intuitive [Caire et al., 2013].

Quelques études [Gorschek et al., 2014; Petre, 2013; Störrle, 2017] ont pu montrer que les schémas UML formels étaient de moins en moins utilisés en industrie et qu'ils laissaient place à des approches de modélisation moins formelles selon les besoins.

Harald Störrle a cherché à comprendre dans quels cas la modélisation était la plus utilisée en industrie. Ses recherches ont pu montrer qu'elle est très présente [Störrle, 2017]. Elle est utilisée de 3 façons différentes : *"informelle"* pour la communication et la cognition, *"semi-formelle"* pour planifier et documenter et *"formelle"* pour les contrats. Parmi les trois, c'est la façon informelle la plus utilisée. Il a pu observer que la modélisation est un élément clé pour les architectes logiciels (avec 91% de réponse positive à la question relative à l'utilité de la modélisation) mais beaucoup moins pour les utilisateurs et intervenants extérieurs (39% de réponse positive). Ces observations ont pu appuyer le manque d'implication des utilisateurs dû au fait qu'ils ne mesurent pas l'importance de ces modélisations.

### 2.3.1 Proposer des notations de diagrammes compréhensibles par les non-experts

Sur base de ces deux constats, Caire et al. proposent une approche radicalement différente [Caire et al., 2013; Genon, 2016] de ce qui a déjà été suggéré par le passé pour rendre la modélisation plus simple et accessible par les non-experts et plus flexible que l'UML qui facilitera la communication. L'idée est d'impliquer les non-experts dans la définition des notations du langage de modélisation. Cette étude a pu montrer que les symboles choisis par les non-experts étaient beaucoup plus simples à être interprétés par de nouveaux intervenants non-experts. Ceci a permis de réduire par 5 le facteur d'erreur lié à l'interprétation d'un modèle. Cette amélioration est due à la transparence des symboles utilisés pour représenter un objet. Par exemple, représenter un utilisateur par un dessin de personnage et non pas par un rectangle avec son nom permet beaucoup plus facilement de lier le symbole à ce qu'il représente (Voir la Figure 2.2).

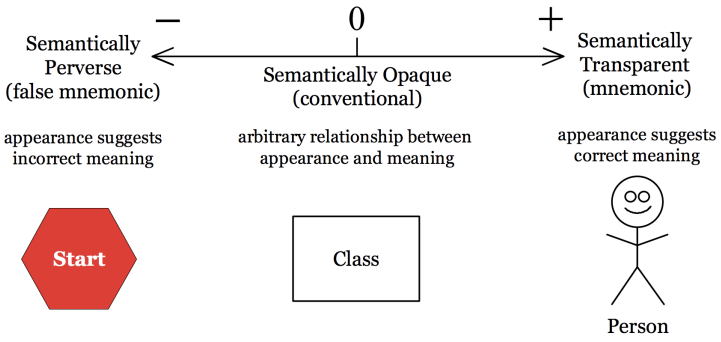


Figure 2.2: Transparence sémantique [Caire et al., 2013])

### 2.3.2 Diagramme de robustesse

Le diagramme de robustesse (DR) est une technique de modélisation introduite par Jacobson en 1987 [Jacobson, 1987] permettant de visualiser le flux d'un scénario et les différentes interactions de celui-ci. À l'aide d'une version modifiée du DR, Rosenberg et Stephens ont prouvé son utilité dans des projets industriels afin de définir des spécifications comportementales [Rosenberg and Stephens, 2007] . El-Attar et Miller l'ont utilisé pour modéliser des tests d'acceptation pour les utilisateurs [El-Attar and Miller, 2010]. L'une des forces de cette modélisation est sa simplicité, elle est facilement compréhensible par un ensemble d'intervenants. À la figure 2.3, nous pouvons retrouver un aperçu des 4 types d'élément et des 2 types de lien composant le diagramme de robustesse. Les différents éléments composant le diagramme de robustesse:

- ***Actor*** : utilisateur à qui fait référence l'action de l'US;
- ***Boundary*** : interface avec laquelle l'acteur interagit avec le système;
- ***Control*** : action que l'acteur accomplit sur le système;
- ***Entity*** : objet du système impacté par l'action de l'utilisateur.

Pour avoir une modélisation correcte des liens, il faut respecter ces contraintes:

- un *Actor* est seulement lié par un simple lien à une/des *Boundary*.
- une *Boundary* est seulement liée par un simple lien à un/des *Actor* et un/des *Control*.
- un *Control* peut être lié par un simple lien à une *Entity* ou par un lien directionnel (→) à un/des *Control* ou une *Boundary*.
- une *Entity* est liée par un simple lien à un/des *Control*.

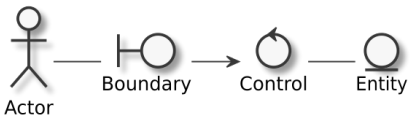


Figure 2.3: Vue d'ensemble du diagramme de robustesse [Gilson et al., 2019b])

## 2.4 Conclusion

En développement logiciel Agile, la méthode de représentation des exigences utilisateurs la plus populaire est celle des histoires d'utilisateur (*"**As a** <acteur>, **I want** <objectif> ,[**so that** <bénéfice>]"*) écrite dans un vocabulaire compréhensible par les différents intervenants. Malheureusement, leur grand nombre rend très compliquée pour chacun la bonne compréhension de l'ensemble du système. L'utilisation d'une visualisation simple et explicite comme le diagramme de robustesse est une option intéressante qui a déjà fait ses preuves dans le monde industriel. Cette visualisation sera un outil clé dans la prise de décision de design souvent très complexe. Un autre élément l'influencera grandement : les différents attributs de qualité présents. Il est donc essentiel de les identifier rapidement au cours du projet.

CHAPTER 3

# TECHNIQUES À BASE DE NLP ET ML POUR LA MODÉLISATION ET L'EXTRACTION

*Dans ce chapitre, nous allons aborder un ensemble de recherches utilisant le ML en NLP afin de convertir du texte en différents types de schéma ou pour extraire des informations permettant d'aider une équipe dans le bon déroulement de ses projets. Dans la première section, nous présenterons différentes approches utilisant le NLP afin de convertir du texte en modèle visuel. Dans la seconde section, nous expliquerons les approches utilisées par différents chercheurs afin de récupérer automatiquement des informations pouvant aider à la prise de décisions de design.*

## 3.1 Utilisation du NLP pour extraire et modéliser du texte en langage naturel

Dans de plus en plus de domaines, la modélisation devient un outil essentiel pour faciliter la compréhension et la communication entre différents intervenants [Störrle, 2017; Moody, 2009]. Le gros problème de celle-ci, est qu'elle reste très souvent une tâche manuelle nécessitant énormément de temps [Osman and Zalhan, 2016]. Souvent, elle consiste simplement à représenter sous forme de diagrammes un ensemble d'informations textuelles. L'avancée des techniques en NLP a poussé certains chercheurs à les utiliser pour proposer des solutions de modélisation automatique à partir de documentations textuelles.

Osman et Zalhan [Osman and Zalhan, 2016] proposent une analyse des différentes approches utilisées pour cette tâche dans la littérature scientifique. Ils présentent un ensemble de techniques NLP employées pour extraire automatiquement des données et un ensemble de méthodes de modélisation visuelle et comparent les propriétés de chacune. Nous décrivons ci-dessous les différentes étapes de plusieurs approches et comment elles ont été évaluées.

### 3.1.1 Modélisation automatique d'un BPMN

Friedrich, Mendling et Puhlmann [Friedrich et al., 2011] nous proposent d'utiliser le NLP pour modéliser des modèles BPMN [OMG, 2011]. La première étape a été de découper le texte en phrases, de les tokenizer, d'identifier des types de mot et créer l'arbre des dépendances.

La seconde étape est de parser l'arbre à l'aide d'un parseur. Celui-ci va extraire l'ensemble des acteurs, des actions et les liens entre ceux-ci sur base des différentes relations grammaticales. Dans BPMN, les relations conditionnelles sont très importantes ; donc, le parseur va les identifier à l'aide des marqueurs de conditions *e.g., "if" ,"then"* et les associera chacune à la modélisation correspondante. Grâce aux informations déjà recueillies, le parseur va être capable d'identifier les actions similaires et les fusionner, ce qui va résoudre le problème d'action décrite dans différentes phrases. Après cela, il va faire la même chose pour les objets, puis pour les acteurs. La dernière tâche du parseur va être d'identifier le flux, qui correspond à l'interaction entre les différents éléments de la modélisation. Pour cela, ils ont utilisé l'hypothèse que les actions étaient décrites de façon séquentielle et donc qu'ils pouvaient se baser sur l'ordre des phrases pour définir le sens du flux. La dernière phase de leur processus de modélisation consiste à utiliser toutes les informations recueillies pour créer leur BPMN. Cette phase a été divisée en 4 étapes : modéliser les noeuds, créer la séquence de flux, retirer les éléments superflus, finaliser les extrémités ouvertes.

Pour évaluer la qualité des BPMN générés par leur technique, ils ont calculé la similarité entre les modèles générés automatiquement et manuellement pour 47 textes de l'industrie. Ils ont obtenu une moyenne de 77% de similarité.

### 3.1.2 Génération automatique de diagrammes UML de séquences depuis des histoires d'utilisateur dans le processus Scrum

M. Elallaoui, K.Nafil et R.Touahni [Elallaoui et al., 2015] proposent une technique permettant la génération automatique de diagrammes UML de séquences depuis des histoires d'utilisateur respectant le template de Cohn. Pour identifier les différents éléments clés (acteur, action et bénéfice), ils vont scanner les US à l'aide de leur algorithme d'extraction basé sur la syntaxe. Ils ont associé l'acteur comme l'émetteur, le bénéfice comme le message et l'objet contenu dans l'action comme le récepteur (s'il n'y en a pas, le récepteur par défaut sera le système). Malheureusement, aucune technique d'évaluation n'a été proposée.

### 3.1.3 Transformation automatique d'histoires d'utilisateur en diagramme UML de Use Case

Un dernier exemple de Elallaoui et al. [Elallaoui et al., 2018] : ils nous proposent une génération de Diagramme UML de Use Case à partir d'US en utilisant le NLP. La première étape est de retirer l'ensemble des mots inutiles des US. Comme dans les exemples précédents, la seconde étape consiste à utiliser un parseur, TreeParser, qui va sélectionner l'ensemble des noms, noms propres, déterminants et verbes importants. Ils ont défini un ensemble de règles basées sur leur type pour transformer ces candidats en acteurs ou en Use Case et de définir leurs relations. Pour évaluer

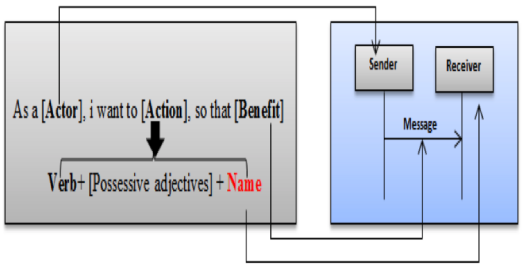


Figure 3.1: Mise en corrspondance entre les parties de l'US et les éléments du diagramme de séquence [Elallaoui et al., 2015])

leur résultat, ils ont comparé la génération automatique à une création manuelle pour 90 US. Ils ont obtenu une *Précision*/*Rappel* de 98%/98% pour les acteurs, 87%/85% pour les Use cases et 87%/85% pour les relations.

## 3.2 Recherche aidant à la prise de décisions de design

Dans cette section, nous allons présenter quelques approches d'automatisation de l'extraction/prédiction d'informations permettant d'aider les développeurs à la prise de décisions de design.

### 3.2.1 Extraction automatique de décisions de design

L'équipe de recherches composée de Bhat et al. [Bhat et al., 2017] est partie du constat qu'il est difficile de faire le choix de design optimal pour un cas donné. Elle a donc voulu proposer une solution permettant d'extraire et de classifier des choix designs basés sur d'anciennes décisions prises dans un cas similaire. Elle propose donc d'utiliser une pipeline NLP composée de deux modèles de classification d'apprentissage automatique. Le premier aura pour but d'identifier les enjeux qui font référence à une décision de design. Le deuxième aura pour objectif de classifier chaque enjeu identifié positif dans la première étape parmi les différents types de décisions design (**"Décision structurelle"**, **"Décision comportementale"** ou **"Décision d'interdiction"**).

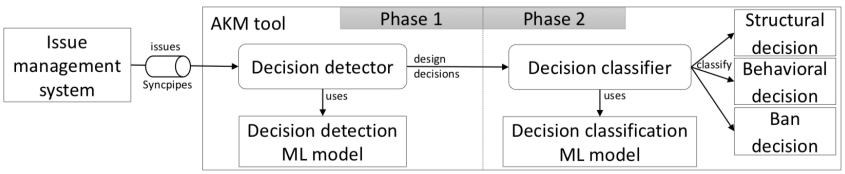


Figure 3.2: Pipeline d'extraction automatique de décisions de design sur base d'enjeux [Bhat et al., 2017])

Pour entraîner leurs deux modèles ML, ils ont dû récupérer une base de données composée de 2,259 enjeux du projet Apache Spark [Apache, 2019] et 420 enjeux du projet Apache Hadoop Common[Apache, 2018a]. Ces eujeux avaient été au préalable sauvegardés par un *issue tracker* nommé Jira[1] *i.e.* est un outil permettant de conserver et centraliser la connaissance sur un projet en mémorisant les différents choix, tâches, etc. réalisés tout au long de celui-ci. L'ensemble des enjeux a été annoté individuellement à la main par deux architectes logiciel ayant plus de 5 ans d'expérience. Ils ont dû déterminer pour chaque enjeu s'il était une décision de design ou non. Si oui, ils ont dû choisir une classe pour cet enjeu. Pour permettre la bonne compréhension des différentes classes par les architectes, une liste de règles les définissant leur a été fournie avant l'annotation. Pour obtenir les meilleurs résultats possible, ils ont évalué un ensemble de différents modèles de classification (SVM, Logistic regression, Naive Bayes, ...) sur leur base de données afin de trouver celui qui serait le plus performant pour chacune des étapes, ce qui leur a permis d'obtenir le résultat de 91.29% d'accuracy avec un classifieur SVM pour l'étape 1 et de 82.79% d'accuracy également avec un SVM pour l'étape 2.

### 3.2.2 Récupération des décisions de design architectural

Pour éviter la perte de connaissances sur des décisions de design antérieures, Arman Shahbazian, Youn Kyu Lee, Duc Le, Yuriy Brun et Nenad Medvidovic ont développé RecovAr [Shahbazian et al., 2018] qui est une solution permettant de récupérer les différentes décisions de design prises au cours d'un projet sur base des enjeux et du versionning du code d'un projet. Une phase de préprocessing avec ARCADE, logiciel permettant d'obtenir l'architecture d'un système, a été appliquée sur différentes versions du code pour obtenir leur architecture statique. Après cette étape, le processus de RecovAr peut être divisé en 3 phases :

- Analyser et identifier les changements entre 2 versions d'une même architecture statique;
- Mapper les enjeux avec des entités de l'architecture;
- Créer un graphe des différentes décisions en rassemblant les changements architecturaux et les enjeux qui s'y rapportent.

La pipeline globale est présentée à la Figure 3.3. Sur base de ce graphe, RecovAr extrait les différentes décisions de design et les associe à du code et à des enjeux. Ils ont évalué leur outil sur 2 projets OpenSource, Hadoop [Apache, 2018a] et Struts [Apache, 2018b]. Ils ont obtenu une *Précision* de 77% et un *Rappel* de 75%.

### 3.2.3 Extraction automatique de tâches de développement dans de la documentation logicielle

Treude et al. [Treude et al., 2015] ont voulu mettre en avant les bienfaits de l'utilisation du NLP dans le développement du logiciel. Ils sont partis du constat qu'en développement du logiciel, la documentation permet rarement à un développeur de trouver aisément l'information dont il a besoin. Pour résoudre ce problème, ils ont développé TaskNav, qui permet à un développeur

[1] https://www.atlassian.com/software/jira

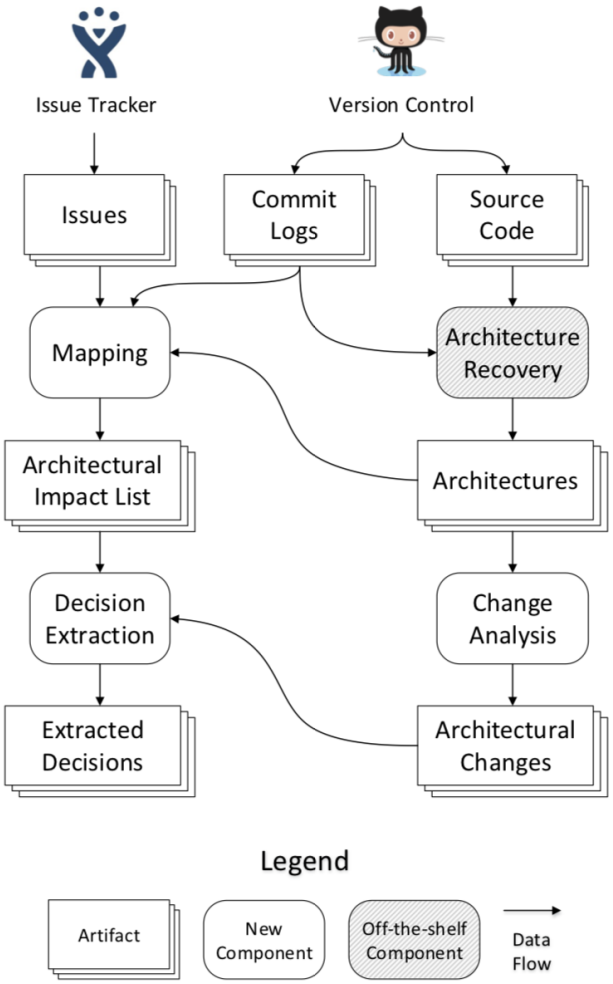


Figure 3.3: Pipeline de récupération de décisions de design par RecovAr [Shahbazian et al., 2018])

d'insérer une question et l'outil lui propose toutes les tâches conceptualisées à partir des actions décrites dans la documentation. Pour arriver à ce résultat à partir de textes non-structurés, ils ont eu recours à un ensemble de techniques NLP. La première étape est le préprocessing pour transformer l'ensemble de la documentation en un texte standard, *e.g.,* retirer les drapeaux HTML, et pouvoir le tokenizer. La deuxième étape consiste à extraire les différentes tâches et les dépendances entre elles à l'aide de l'arbre des dépendances. Pour identifier une tâche, ils vont considérer chaque verbe lié à un objet comme un potentiel candidat et ajouter à chacun son contexte. Pour éviter d'avoir des tâches n'ayant rien à voir avec le développement, ils vont ajouter un filtre retirant toutes les tâches n'ayant pas leur verbe principal dans une liste de verbes définie à la main.

Afin d'évaluer les tâches et les concepts extraits par TaskNav, ils ont fait appel à 8 développeurs

de la société Xprima (dont 4 seniors) pour le tester sur l'un de leurs projets. Ils ont sélectionné aléatoirement 196 tâches sur les 1053 extraites et ils les ont réparties entre les différents testeurs (50% des tâches ont été attribuées à deux différents testeurs simultanément). Pour chaque tâche reçue, le testeur a dû répondre à la question "S'agit-il d'une tâche (ou d'une sous-tâche) pour les intégrateurs HTML ou les développeurs travaillant sur[projet] ?". Ils ont obtenu un "oui" pour 50% des tâches extraites. La même question a été posée pour les 100 concepts choisis aléatoirement parmi les 131 extraits et ils ont obtenu 66% de "oui". La même méthode a été utilisée pour Django sur deux développeurs cette fois. Pour les 36 tâches sélectionnées sur 1209, ils ont obtenu 44% de "oui" et pour les 25 concepts extraits, ils ont obtenu 62% de "oui". En comparant les différentes réponses obtenues par les deux développeurs, ils ont pu observer que dans 47% des cas pour les tâches et 33% pour les concepts, l'un disait "oui" et l'autre "non".

### 3.2.4 Classification automatique d'exigences non-fonctionnelles à partir de commentaires d'utilisateurs augmentés d'une application mobile

Mengmeng Lu et Peng Liang [Lu and Liang, 2017] partent du constat qu'il y a un très grand nombre de commentaires d'utilisateurs à propos d'application mobile qui, si ils étaient bien utilisés, pourraient aider largement les développeurs à créer un produit correspondant aux attentes de leurs utilisateurs. Ils ont comme objectif de proposer une solution qui pourra classifier automatiquement les commentaires d'utilisateurs parmi les différentes exigences non-fonctionnelles (*reliability*, *usability*, *portability*, et *performance*).

Pour tester leur approche, ils ont récupéré une base de données composée de 6696 commentaires d'utilisateurs à propos de iBook [Apple, 2010] sur la plateforme Apple App Store et 4400 à propos de WhatsApp [Facebook, 2014] sur le Google Play. Pour les deux applications, 2000 commentaires ont été sélectionnés aléatoirement et ont été annotés manuellement par 3 chercheurs. Sur base de ces 4000 données annotées, ils ont d'abord effectué une phase de préprocessing (retirer les mots d'arrêt, lemmatisation, découpe en phrases,...). Ensuite, ils augmentent la taille de chaque phrase en ajoutant une liste de taille fixe composée des mots les plus similaires à ceux qui composent cette phrase dans la base de données selon Word2Vec. Ils utilisent cette technique pour résoudre le problème de taille des phrases en ajoutant des mots venant du même domaine. Pour obtenir les meilleurs résultats possible, ils ont testé différents encodages pour leurs phrases (BOW, CHI, TF-IDF [Zhang et al., 2010; Bornstein, 2018; Forman, 2003]) et différents modèles de classification (Naïve Bayes, J48 et Bagging [Zhang and Li, 2007; Bashir and Chachoo, 2017; Tran et al., 2017]). Voici leurs résultats pour une validation croisée en 10 fois, 10-fold cross-validation :Leur meilleur *F1-Score*, 71% pour une validation croisée en 10 fois, 10-fold cross-validation en utilisant l'encodage AUR-BoW avec le modèle Bagging.

## 3.3 Conclusion

La modélisation est un outil facilitant la compréhension et la communication entre les différents intervenants. Son coût en temps, souvent très important, a incité plusieurs équipes de recherches

à proposer un moyen de modéliser de la documentation textuelle automatiquement. Parmi les différentes approches proposées, nous pouvons observer une pipeline récurrente en trois phases pour la modélisation de textes :

- une première de préprocessing dans laquelle le texte est découpé en tokens. Chaque token est associé à son POS/TAG et l'arbre des dépendances est créé.
- une seconde de parsing de l'arbre des dépendances afin d'identifier les éléments clefs et les relations qu'ils partagent pour la modélisation finale.
- une dernière de modélisation des objets et des relations identifiés par le parseur.

La prise de décision de design étant une tâche complexe, plusieurs chercheurs ont proposé des techniques permettant d'extraire et/ou d'identifier des informations clés à partir de documentation textuelle à l'aide du NLP et du ML. Certains d'entre eux se concentrent sur la récupération et l'identification de décisions de design sur base d'enjeux. D'autres essaient d'extraire des tâches et des concepts non documentés depuis de la documentation logicielle. D'autres encore ont proposé d'utiliser les commentaires d'utilisateurs afin d'analyser l'importance des différentes exigences non fonctionnelles d'une application.

Dans le Chapitre 4, au vu de l'efficacité des différentes approches présentées, nous allons tenter d'utiliser le ML pour identifier les histoires d'utilisateur faisant référence à des attributs de qualité. Nous comparerons une approche simple utilisant un seul modèle à la technique en deux étapes proposée par Bath. Dans les Chapitres 5 et 6, nous allons nous inspirer de cette pipeline en trois étapes pour la modélisation automatique de texte.

# Part II

# Contribution

CHAPTER 4

# EXTRACTION AUTOMATIQUE D'ATTRIBUTS DE QUALITÉ DANS DES HISTOIRES D'UTILISATEUR

*Dans ce chapitre, nous allons aborder l'ensemble de nos recherches sur la détection d'US contenant un attribut de qualité et l'identification de son type par le biais de différentes techniques de ML. Dans la première section, nous expliquerons la base de données utilisée et notre méthode de labellisation. Dans la seconde, nous présenterons les résultats obtenus avec la librairie SpaCy pour l'identification des QAs liés à des US. Dans la troisième, nous aborderons lceux obtenus pour des expérimentations similaires à la section précédente avec d'autres techniques d'encodage et de classification. Dans la dernière, nous comparerons nos résultats et nous les discuterons.*

## 4.1 Introduction

Comme nous l'avons expliqué précédemment, les attributs de qualité ont une importance cruciale dans la prise de décision de design [Bass et al., 2002]. Il est donc nécessaire de les détecter rapidement et d'analyser leur importance dans le projet. Nous pensons donc qu'un outil capable de les détecter automatiquement pourrait aider grandement les architectes logiciel limités par leurs capacités cognitives en leur permettant d'analyser des backlogs de plus en plus importants.

Au vu des projets présentés au Chapitre 3 utilisant le ML en NLP afin de détecter des décisions de design, nous pensons qu'il est possible d'entraîner un modèle de ML à la détection des US faisant référence à des QAs et l'identification du type de ceux-ci. Au vu de la complexité de notre tâche, nous avons comme première intuition de nous tourner vers des modèles plus complexes comme l'apprentissage profond. Pour confirmer notre intuition, nous avons comparé les résultats

obtenus par SpaCy à ceux obtenus par d'autres modèles de ML (LinearSVC[1], LogisticRegression[2] et ComplementNB[3]) avec différents encodages (TF-IDF, Glove et BERT). Ayant pour seul but de prouver la faisabilité de nos tâches, nous n'avons pas encore essayé d'optimiser les méta-paramètres des modèles que nous avons entraînés. De plus, nous voulons offrir une vue d'ensemble sur l'importance des différents QAs dans un backlog à l'aide d'un classement créé sur base du nombre d'US préditent comme étant liées à un type de QA.

## 4.2 Base de données

### 4.2.1 Source des données

Pour pouvoir entraîner un modèle de classification supervisée, il est nécessaire d'avoir un ensemble de données labellisées avec les différentes classes possible. Nous avons donc utilisé une base de données OpenSource composée de 1,675 US réparties entre 22 backlogs de projets industriels ou universitaires (Table 6.3). Celle-ci a été créée par F. Dalpiaz [Dalpiaz, 2018]. Chaque US a donc été écrite par des personnes du métier. Il est évident que cette base de données n'est pas suffisante pour représenter l'ensemble des US du monde réel. Toutefois, il est difficile de pouvoir affirmer avoir trouvé une base de données couvrant l'ensemble des cas. Nous pensons donc que ce jeu de données est suffisant pour entamer de premières expérimentations.

### 4.2.2 Processus de labellisation

Suite à cette collecte, les données ont dû être labellisées. Cette labellisation a été réalisée manuellement par les Professeurs F. Gilson et M. Galster de l'Université de Canterbury à Christchurch. Pour ce faire, ils ont d'abord identifié un nombre restreint d'attributs de qualité qu'ils ont considérés comme les plus impactants sur les différentes décisions de design d'un projet (*compatibility*, *security*, *usability*, *maintainability*, *performance*, *portability*, *reliability*)[ISO/IEC, 2010]. Ensuite, chacun indépendamment, a identifié pour chaque US si celle-ci contenait au moins un QA et si oui, lesquels (au maximum deux par US). Finalement, ils ont unifié leur labellisation et ont discuté les différentes contradictions pour obtenir le moins d'erreur possible. Cette approche a été considérée comme suffisante car deux personnes ont réalisé le travail indépendamment et elles jouissaient toutes les deux d'une grande expérience dans ce domaine étant donné leurs précédents travaux de recherches.

### 4.2.3 Résultats de cette labellisation

Le Tableau 4.2 présente le nombre d'occurrences pour chaque QA.

La Figure 4.1 nous montre les 15 mots ayant le plus d'occurrences parmi l'ensemble des US liées à au moins un QA et celles ne faisant référence à aucun QA. Nous pouvons observer que pour les deux catégories, ce sont des mots génériques. Nous ne pouvons donc rien en déduire.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html

Table 4.1: Noms et descriptions des différents backlogs

| Backlog | Description |
|---|---|
| FederalSpending | Web platform for sharing US government spending data |
| Loudoun | Land management system for Loudoun County, Virginia |
| Recycling | Online platform to support waste recycling |
| OpenSpending | Website to increase transparency of government expenses |
| FrictionLess | Platform for obtaining insights from data |
| ScrumAlliance | First version of the Scrum Alliance website |
| NSF | New version of the NSF website |
| CamperPlus | App for camp administrators and parents |
| PlanningPoker | First version of the PlanningPoker.com website |
| DataHub | Platform to find, share and publish data online |
| MIS | Management information system for Duke University |
| CASK | Toolbox to for fast and easy development with Hadoop |
| NeuroHub | Research data management portal |
| Alfred | Personal interactive assistant for active aging |
| BadCamp | Conference registration and management platform |
| RDA-DMP | Software for machine-actionable data management plans |
| ArchiveSpace | Web-based archiving information system |
| UniBath | Institutional data repository for the University of Bath |
| DuraSpace | Repository for different types of digital content |
| RacDam | Software for archivists |
| CulRepo | Content management system for Cornell University |
| Zooniverse | Platform that allows anyone to help with research tasks |

Table 4.2: Nombre d'occurrences par attribut de qualité dans les US labellisées

| Attribut de qualité | Nombre |
|---|---|
| *compatibility* | 165 |
| *security* | 97 |
| *usability* | 80 |
| *maintainability* | 57 |
| *performance* | 31 |
| *portability* | 25 |
| *reliability* | 28 |
| **Nombre total d'US faisant référence à un QA** | **434** |
| US ne faisant référence à aucun QA | 1241 |
| **Nombre total d'US** | **1,675** |

## 4.3 Classification par SpaCy

Pour pouvoir obtenir des résultats prometteurs rapidement, nous avons choisi d'utiliser SpaCy. Cette librairie offre le meilleur compromis que nous avons pu trouver entre la simplicité d'utilisation et une implémentation de modèles performants. Une utilisation de cette librairie dans le cadre d'un projet antérieur a pu confirmer ce choix [Gilson and Irwin, 2018].

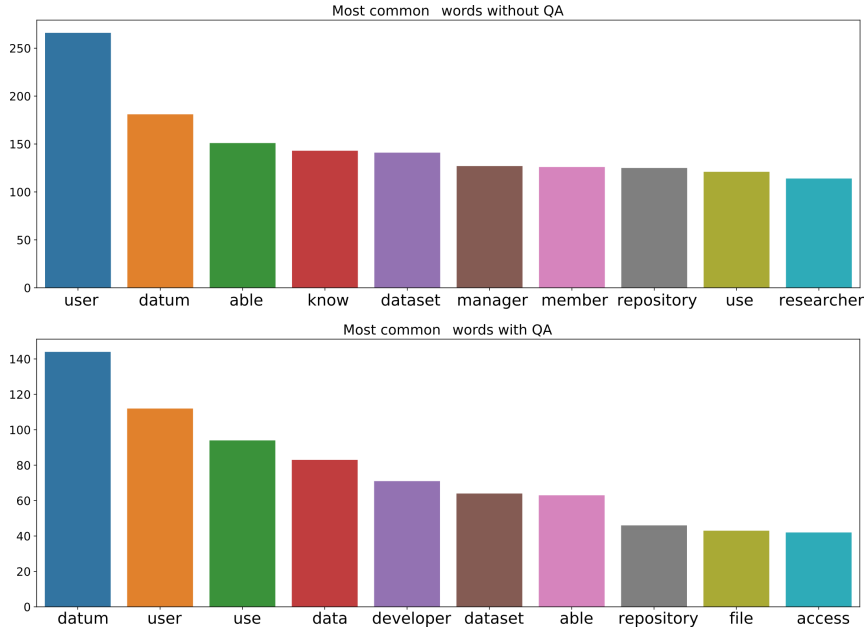


Figure 4.1: 15 mots les plus fréquents pour les US liées à aucun QA et à au moins un QA [Gilson et al., 2019a])

### 4.3.1 Méthode

En apprentissage profond, il est devenu très commun d'utiliser des modèles pré-entraînés pour accélérer l'obtention de nos résultats et les améliorer. Nous avons donc utilisé l'un des modèles pré-entraînés par SpaCy (*en_core_web_lg*[4]) et nous allons l'entraîner pour notre tâche. Pour cela, nous allons donc diviser notre base de données en une d'entraînement (2/3) et une de test (1/3). Nous avons veillé à ne jamais avoir d'US commune aux 2 ensembles pour ne pas biaiser nos résultats. Ayant une base de données très mal équilibrée (26% d'US liées à un QA), nos modèles vont très rarement prédire qu'une US est en lien avec un QA. Pour résoudre ce problème, nous avons décidé de retirer aléatoirement une partie des US ne faisant pas référence à un QA.

Pour identifier si une US est liée à un QA ou non, nous garderons un nombre d'US sans QA égal à celui ayant un QA (50% avec QA et 50% sans QA). Pour l'identification des QA liés à une US, nous avons décidé arbitrairement de retirer 77% des US n'ayant pas de QA. Deux manières différentes ont été utilisées pour obtenir les résultats que nous présenterons.

Pour la première, nous avons entraîné un seul modèle sur les données d'entraînement et nous l'avons évalué sur celles de test. Nous utiliserons la même découpe des données pour éviter une évaluation biaisée par un changement. Pour rendre ces données les plus pertinentes possible, nous avons réparti pour chaque type de QA les US qui y faisaient référence de telle manière à avoir 33% d'entre elles pour le test et de 77% pour l'entraînement. Cette contrainte a pour but d'éviter la

[4]https://spacy.io/models/en

sous-représentation d'un QA dans un des deux ensembles.

La seconde est une validation par *k-fold cross-validation* et nous choisirons un k=10 qui est considéré comme la meilleure valeur par défaut pour cette tâche. Lors de l'évaluation d'un fold, nous nous assurons qu'aucun élément de celui-ci ne se trouve dans les autres pour éviter un biais. Vu la faible représentation de certains QA, nous pourrons déplorer l'absence d'un ou plusieurs types de QA dans certains folds. Pour chaque expérimentation, nous préciserons si nous avons utilisé la *k-fold cross-validation*.

Pour évaluer nos modèles, nous aurons recours aux métriques de *Précision*, *Rappel* et *F1-Score* car nous les considérons comme les plus intéressantes pour notre problème. L'absence d'un/plusieurs QA dans un fold entraînera des valeurs erronées. Par exemple, notre modèle prédit qu'aucune US ne possède un QA et que cela est correct. Nous aurons toutes nos métriques à zéro malgré des prédictions correctes. Pour remédier à cela, nous avons décidé que si nous n'avons aucune occurrence d'une classe et qu'aucune n'est prédite, nous aurons une *Précision* = 1, *Rappel* = 1 et un *F1-Score* = 1 car notre modèle n'a pas commis de faux positif. Lorsque nous parlons du score obtenu par un modèle, nous utiliserons toujours le meilleur obtenu pour *F1-Score* (la *Précision* et le *Rappel* correspondent à ceux utilisés pour calculer le *F1-Score*) sur l'ensemble des époques. Lors d'une validation par *cross-validation*, les résultats présentés correspondent à la moyenne des meilleurs résultats obtenus par *fold*.

Pour chacune de nos expérimentations, nous avons décidé d'entraîner notre modèle sur 100 époques. Une époque correspond à un cycle d'entraînement complet où l'ensemble des données d'entraînement auront été utilisées. Le modèle sera évalué à chaque époque pour connaître sa courbe d'apprentissage. Pour pouvoir l'améliorer, les données d'entraînement seront découpées en petits ensembles (4 à 32 US). Pour chaque époque, tous les petits ensembles d'US seront utilisés un par un pour entraîner le modèle. La taille de ceux-ci grandira au fur à mesure du nombre d'époques. Cette technique permet d'accélérer la vitesse d'apprentissage du modèle et de réduire le risque d'être bloqué dans un optimum local.

## 4.4 Résultats

### 4.4.1 Identification des US faisant référence à un attribut de qualité

En utilisant la méthode décrite précédemment, nous avons entraîné un modèle à identifier si une US fait référence ou non à un QA. A la Figure 4.2, nous pouvons retrouver la courbe d'apprentissage de ce modèle.

Au vu de notre base de données limitées, les performances obtenues par notre classifieur sont raisonnables avec une *Précision* de 0.65, un *Rappel* de 0.67 et un *F1-Score* de 0.66.

En nous inspirant de la pipeline proposée par M. Bhat et son équipe [Bhat et al., 2017], nous avons entraîné un modèle SpaCy à identifier le/les type de QA auquel est liée une US. Nous avons donc entraîné et testé ce modèle uniquement sur des US liées à au moins un QA. Il ne faut pas oublier que dans la classification de M. Bhat [Bhat et al., 2017], chaque objet n'avait qu'une et une

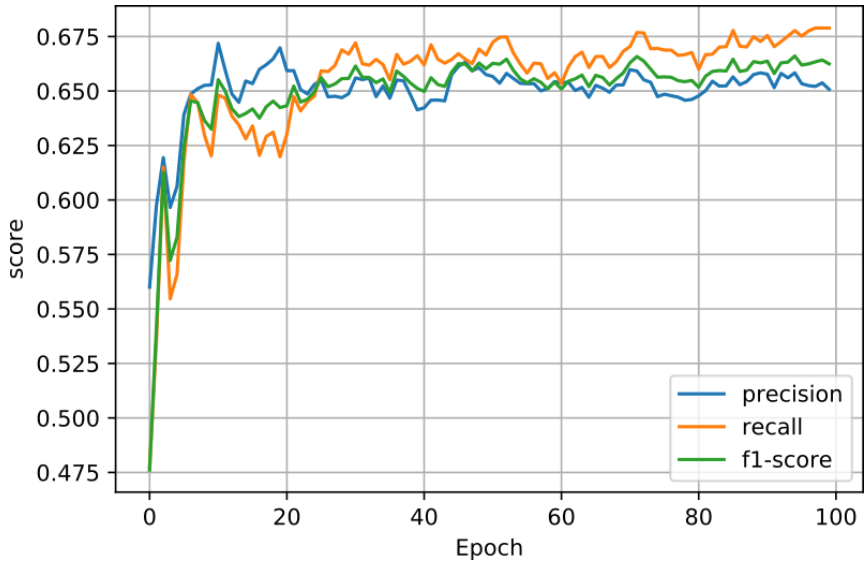


Figure 4.2: Courbe d'apprentissage du modèle QA/No QA [Gilson et al., 2019a]

seule classe possible, contrairement à notre cas où une US peut être liée à plusieurs QAs. La Table 4.3 présente les résultats obtenus pour une *10-fold cross-validation*.

Table 4.3: Résultats pour la *k-fold cross-validation* par QA uniquement sur des US liées à au moins un QA

| **QA** | $prec$ | $rec$ | $f_1$ |
|---|---|---|---|
| *performance* | 0.62 | 0.42 | 0.47 |
| *compatibility* | 0.74 | 0.69 | 0.71 |
| *usability* | 0.66 | 0.46 | 0.52 |
| *reliability* | 0.2 | 0.08 | 0.11 |
| *security* | 0.77 | 0.64 | 0.69 |
| *maintainability* | 0.69 | 0.56 | 0.58 |
| *portability* | 0.6 | 0.38 | 0.45 |
| ***Global*** | **0.71** | **0.53** | **0.6** |

### 4.4.2 Un modèle pour tous ou tous pour un

Comme expliqué précédemment, deux approches permettent de résoudre un problème de classification multi-label. Nous allons donc comparer les résultats d'un modèle global entraîné à identifier tous les QA liés à une US et un ensemble de modèles, chacun capable de classifier pour un type de QA si l'US y fait référence ou non. Comme expliqué dans notre méthode, nous allons utiliser pour l'entraînement 2/3 des données et 1/3 pour l'évaluation. La même répartition sera utilisée pour entraîner chaque modèle afin d'éviter un biais. Cette évaluation ayant pour but d'être rapide,

nous n'avons pas utilisé de *k-fold cross-validation*. Dans la Table 4.4, vous pouvez retrouver la comparaison des performances obtenues par les deux stratégies.

Table 4.4: Comparaison du modèle global par rapport à un modèle par QA [Gilson et al., 2019a]

| | **Modèle Global** | | | **Modèle Spécialisé** | | |
|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.99 | 0.44 | 0.61 | 0.71 | 0.55 | 0.62 |
| *compatibility* | 0.66 | 0.56 | 0.60 | 0.67 | 0.54 | 0.60 |
| *usability* | 0.76 | 0.38 | 0.51 | 0.49 | 0.34 | 0.40 |
| *reliability* | 0.49 | 0.19 | 0.28 | 0.99 | 0.09 | 0.18 |
| *security* | 0.66 | 0.41 | 0.50 | 0.65 | 0.44 | 0.52 |
| *maintainability* | 0.74 | 0.35 | 0.47 | 0.74 | 0.35 | 0.47 |
| *portability* | 0.99 | 0.19 | 0.33 | 0.66 | 0.19 | 0.30 |

Malgré des scores très proches, le modèle global est souvent légèrement meilleur. Nous avons donc opté pour l'approche globale qui sera utilisée pour le reste de nos expérimentations.

### 4.4.3 Evaluation par Cross-Validation

Pour rendre nos résultats plus pertinents, nous avons évalué l'approche globale par *10-fold cross-validation*. Vous pouvez retrouver à la Figure 4.3 la courbe d'apprentissage *Global* de notre modèle. Pour calculer ces valeurs *Global*, nous utiliserons les TP, FP, TN, FN globaux qui correspondent aux sommes des TP, FP, TN, FN de tous les QAs. Par exemple, nous avons deux QA différents classifiés par notre modèle. Le premier obtient $TP = 3$ et le second $TP = 6$, nous aurons donc le $TP - Global = 6 + 3 = 9$

Nous pouvons retrouver à la table 4.5, les meilleures performances par QA et pour la valeur *Global* par *10-fold cross-validation*. Nous pouvons voir que pour les valeurs *Global*, nous avons une *Précision* de 0.74 , un *Rappel* de 0.41 et un *F1-Score* de 0.53. Nous observons que l'éventail des *F1-Score* par QA est très large, il va de 0.18 pour le plus mauvais jusqu'à 0.63 pour le meilleur. On peut donc constater que *compatibility*, *maintainability* et *security* sont classifiés de façon correcte à un niveau acceptable avec un *F1-Score* proche de 0.6. Pour *performance* et *portability*, la classification est plus médiocre mais pas nécessairement inacceptable avec un *F1-Score* aux alentours de 0.4. Finalement, nous constatons que notre classifieur est extrêmement mauvais pour identifier *usability* et *reliability* avec un *F1-Score* inférieur à 0.3.

Si nous comparons les résultats obtenus pour le classifieur QA/non QA à *Global* de la Table 4.5, nous pouvons constater que le *F1-Score* est légèrement inférieur mais la différence entre le *Rappel* et la *Précision* de chacun est bien plus grande. Nous pouvons expliquer cela par l'équilibrage des données car le classifieur QA/non QA a été entraîné et testé sur des données totalement balancées (50% de QA et 50% de non QA) contrairement au modèle *Global* ayant une proportion très réduite de chaque QA par rapport à l'ensemble. L'approche de classification QA/non QA prédira plus d'US ayant un QA mais cela causera un plus grand nombre de FP.

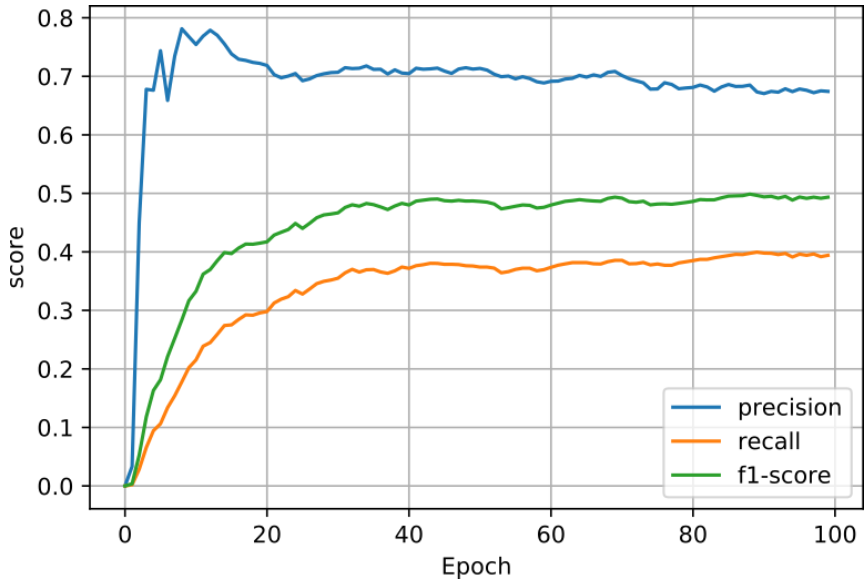


Figure 4.3: Courbe d'apprentissage *Global* du modèle SpaCy par époque avec une *10-fold cross-validation* [Gilson et al., 2019a]

Table 4.5: Résultats pour la *k-fold cross-validation* par QA [Gilson et al., 2019a]

| **QA** | $prec$ | $rec$ | $f_1$ |
|---|---|---|---|
| *performance* | 0.57 | 0.31 | 0.39 |
| *compatibility* | 0.77 | 0.55 | 0.63 |
| *usability* | 0.61 | 0.20 | 0.29 |
| *reliability* | 0.29 | 0.14 | 0.18 |
| *security* | 0.66 | 0.50 | 0.57 |
| *maintainability* | 0.77 | 0.50 | 0.59 |
| *portability* | 0.56 | 0.39 | 0.44 |
| ***Global*** | **0.74** | **0.41** | **0.53** |

### 4.4.4 Seuil de décisions

Nous avons aussi exploré rapidement l'influence du seuil de décision sur les résultats de nos modèles. Le seuil de décision correspond à la valeur minimale nécessaire lors d'une prédiction pour que celle-ci soit considérée comme Positive et si elle est inférieure, la prédiction sera considérée comme Négative. Dans la Table 4.6, nous comparons les résultats obtenus pour un modèle entraîné avec un seuil de 0.5 (celui utilisé par défaut pour tous les résultats précédents) et de 0.6 en utilisant une *10-fold cross-validation*.

On peut constater que la modification de celui-ci peut apporter des améliorations ou des diminutions du score en fonction du QA. Pour *performance*, *compatibility*, *usability* et *security*, leurs scores s'améliorent, pour certaines, de façon significative. A contrario, celui de *reliability*,

Table 4.6: Comparaison des performances obtenues en fonction de différents seuils de décisions pour une *10-fold cross-validation* [Gilson et al., 2019a]

| | **0.5 seuil** | | | **0.6 seuil** | | |
|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.57 | 0.31 | 0.39 | 0.92 | 0.61 | 0.69 |
| *compatibility* | 0.77 | 0.55 | 0.63 | 0.79 | 0.55 | 0.64 |
| *usability* | 0.61 | 0.20 | 0.29 | 0.56 | 0.22 | 0.31 |
| *reliability* | 0.29 | 0.14 | 0.18 | 0.19 | 0.06 | 0.09 |
| *security* | 0.66 | 0.50 | 0.57 | 0.83 | 0.61 | 0.69 |
| *maintainability* | 0.77 | 0.50 | 0.59 | 0.74 | 0.36 | 0.47 |
| *portability* | 0.56 | 0.39 | 0.44 | 0.54 | 0.27 | 0.35 |
| ***Global*** | **0.74** | **0.41** | **0.53** | **0.73** | **0.42** | **0.53** |

*maintainability* et *portability* diminue. Malgré ces changements, le résultat *Global* reste presque inchangé.

Pour mieux comprendre les impacts de la modification du seuil, nous avons extrait l'ensemble des prédictions modifiées causées par un nouveau seuil de décisions sur les 10 fold du modèle par défaut (0.5).

Malheureusement, l'utilisation de deux modèles entraînés séparément et ayant comme simple différence leur seuil de décisions engendre un biais dans nos résultats. L'entraînement d'un même modèle SpaCy réalisé séparément trouvera des optimaux différents et donc leurs résultats varient indépendamment des seuils. L'architecture de notre code ne nous permettant pas d'obtenir la comparaison de seuils différents pour les résultats optimaux d'un même modèle, nous avons dû nous contenter de récupérer les prédictions obtenues après les 100 époques (cela implique que les observations suivantes ne sont pas réalisées sur le modèle obtenant les meilleurs résultats) et d'identifier ceux impactés par le passage de 0.5 à 0.6 pour chacun des folds. Nous pouvons retrouver ces observations dans la Table 4.7. Celle-ci montre le nombre de prédictions devenues Vraies et Fausses par QA par Fold.

Table 4.7: Nombre de prédictions devenues vraies et fausses en passant du seuil 0.5 à 0.6

| **Id fold** | $NbVrai$ | $NbFausse$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 3 | 0 | 0 |
| 4 | 3 | 1 |
| 5 | 1 | 0 |
| 6 | 3 | 1 |
| 7 | 1 | 1 |
| 8 | 1 | 1 |
| 9 | 2 | 0 |
| **Total** | **12** | **5** |

D'abord, nous pouvons observer que l'ensemble des folds n'a pas été affecté. Ensuite, nous constatons des prédictions corrigées pour les folds [2, 4, 5, 6, 7, 8, 9] et de nouvelles erreurs pour les folds [4, 6, 7, 8]. Globalement, les résultats sont meilleurs mais ceux-ci varient selon les QA. Pour *performance*, *usability* et *maintainability* nous avons uniquement des améliorations. Pour *security* et *compatibility*, nous avons des erreurs et des corrections mais le nombre de corrections est supérieur. On peut constater que *reliability*, contrairement aux autres QAs, aura de moins bons résultats. Ces observations confirment notre hypothèse de la première expérimentation. Pour certains QA, la modification du seuil améliore les performances et pour d'autres, elle les réduit.

### 4.4.5 Taille de la base de données

En ML et particulièrement en apprentissage profond, la taille de la base de données sur laquelle un modèle s'entraîne a une importance cruciale. Pour savoir si augmenter le nombre d'US utilisées pour l'entraînement permettra d'obtenir de meilleurs résultats, nous avons décidé d'observer l'évolution de nos différentes métriques d'évaluation en fonction de la quantité de données d'entraînement. Nous avons donc entraîné un modèle avec 100, 200, 300, 400, 500 et 600 US. Leur découpe en bloc de 100 US a été définie aléatoirement avant l'expérimentation. Pour chacun des modèles, nous ajouterons les 100 US suivantes à celles utilisées pour l'entraînement du précédent. Nous pouvons retrouver à la Figure 4.4 l'évolution de nos résultats en fonction de la taille de la base de données d'entraînement. Chacun des modèles a été évalué sur les mêmes données.

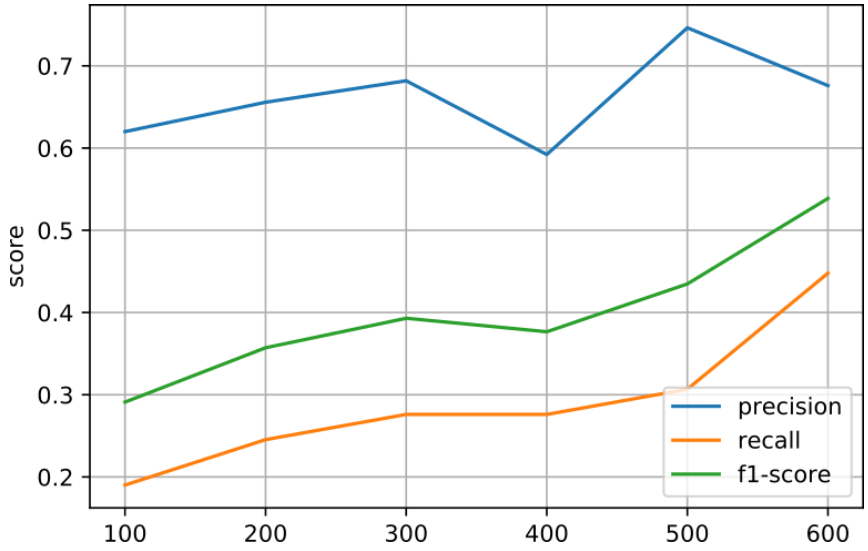


Figure 4.4: Courbe d'apprentissage du modèle SpaCy en fonction du nombre de données d'entrainement [Gilson et al., 2019a]

Nous pouvons observer sur la Figure 4.4 que nos résultats progressent en fonction de notre nombre de données. La progression ne stagne pas. Nous pouvons donc déduire que nous n'avons pas encore atteint le seuil d'apprentissage maximum. Ce seuil représente le nombre de données à

partir duquel offrir plus de données d'entraînement ne permettra plus d'améliorer les performances de notre modèle. Nous croyons donc qu'augmenter le nombre de données labellisées permettra d'améliorer significativement nos résultats.

### 4.4.6 Evaluation par backlog

Cette dernière expérimentation sur nos modèles SpaCy veut montrer s'il est possible ou non de prédire une vue d'ensemble fiable des QAs présents dans un backlog. Pour maximiser les données d'entraînement pour chaque backlog, nous avons entraîné un modèle SpaCy sur une base de données composée des US provenant des autres backlogs. Comme précédemment, nous n'utilisons que 30% des US n'étant pas liées à un QA pour chaque modèle. Le reste des paramètres est similaire aux expérimentations passées. Une fois le modèle entraîné, nous l'utiliserons pour prédire les QAs présents dans chaque US du backlog pour lequel il a été créé.

Dans la table 4.8, nous pouvons retrouver nos différents résultats par backlog. Pour chaque backlog, nous aurons l'ordre d'importance des QA observés selon une approche manuelle et selon les prédictions de notre modèle. L'ordre d'importance d'un QA a été calculé en fonction du nombre d'US liées à ce QA. Pour évaluer les différences entre notre annotation manuelle et automatique, nous avons utilisé trois métriques différentes :

- **Missed** : le nombre de types de QA qui auraient dû être présents dans les prédictions du modèle selon l'annotation manuelle.
- **Additional** : le nombre de types de QA qui n'auraient pas dû être prédits comme présents par notre modèle selon l'annotation manuelle.
- **Swaps** : est le nombre d'inversions de QA dans la liste des prédictions pour obtenir le même ordonnancement que l'annotation manuelle.

Par exemple, la séquence sm = $\{P, C, R\}$ de l'identification manuelle et sa = $\{P, R\}$ de l'identification automatique conduirait à une comparaison des séquences sm = P, C, R et sa = $\{P, R, \epsilon\}$. (où $\epsilon$ indique la chaîne vide). Nous aurions besoin d'un échange ("swap") entre C et R pour déplacer C à la position $\epsilon$ en sm. Le nombre total de swaps requis dans l'exemple serait alors de 1.

Dans la colonne "Difference" de la Table 4.8, nous observons que pour seulement deux backlogs (FrictionLess et NSF), la prédiction automatique détecte des QAs inexistants dans le backlog. A contrario, seulement quatre backlogs (ScrumAlliance, NeuroHub, DuraSpace et RacDam) auront l'ensemble de leurs QAs prédits. Sur base de ces deux observations, nous pouvons donc constater qu'en règle générale, le modèle prédira un sous-ensemble des QAs réellement présents dans un backlog.

Pour ce qui est de la similarité entre l'ordre d'importance des QAs par backlog, nous trouverons peu de différences et celles-ci seront souvent causées par les QAs non prédits. Le nombre de Swaps (voir la colonne "Swaps" de la Table 4.8) reste souvent très faible à l'exception des backlogs NSF et CASK pour lequel il atteint la valeur de 3. Pour NSF, nous pouvons expliquer cette valeur par le faible nombre d'occurrences de chaque QA (deux pour chacun).

Table 4.8: Séquences classées par ordre d'importance des QAs [Galster et al., 2019]

| **Backlog** | **Sequences** | **Difference** | **Swaps** |
|---|---|---|---|
| FederalSpending (manual)<br>FederalSpending (automatic) | $\{M, C, PF, S\}$<br>$\{C\}$ | Missed: 3<br>Additional: 0 | 1 |
| Loudoun (manual)<br>Londoun (automatic) | $\{C, S\}$<br>$\{C\}$ | Missed: 1<br>Additional: 0 | 0 |
| Recycling (manual)<br>Recycling (automatic) | $\{C, S, M, PT\}$<br>$\{S, C\}$ | Missed: 2<br>Additional: 0 | 1 |
| OpenSpending (manual)<br>OpenSpending (automatic) | $\{C, M, S, PT\}$<br>$\{C,\ S\}$ | Missed: 2<br>Additional: 0 | 1 |
| FrictionLess (manual)<br>FrictionLess (automatic) | $\{C, PF, R, M\}$<br>$\{C, PF, S, M\}$ | Missed: 1<br>Additional: 1 | 1 |
| ScrumAlliance (manual)<br>ScrumAlliance (automatic) | $\{S, C\}$<br>$\{S,\ C\}$ | Missed: 0<br>Additional: 0 | 1 |
| NSF (manual)<br>NSF (automatic) | $\{C, M, PT\}$<br>$\{M, S, C\}$ | Missed: 1<br>Additional: 1 | 3 |
| CamperPlus (manual)<br>CamperPlus (automatic) | $\{S\}$<br>None | Missed: 1<br>Additional: 0 | 0 |
| PlanningPoker (manual)<br>PlanningPoker (automatic) | $\{C, PF, S\}$<br>$\{S, C\}$ | Missed: 1<br>Additional: 0 | 2 |
| DataHub (manual)<br>DataHub (automatic) | $\{C, R, PT, S, M\}$<br>$\{C\}$ | Missed: 4<br>Additional: 0 | 0 |
| MIS (manual)<br>MIS (automatic) | $\{S, C, M, PT, R\}$<br>$\{C, S\}$ | Missed: 3<br>Additional: 0 | 1 |
| CASK (manual)<br>CASK (automatic) | $\{M, R, PT, C\}$<br>$\{C, M\}$ | Missed: 2<br>Additional: 0 | 3 |
| NeuroHub (manual)<br>NeuroHub (automatic) | $\{C, S, PT, M, R, PF\}$<br>$\{C, S, M, PT, PF, R\}$ | Missed: 0<br>Additional: 0 | 2 |
| Alfred (manual)<br>Alfred (automatic) | $\{C, S, M, PT, PF\}$<br>$\{C, S, M, PT\}$ | Missed: 1<br>Additional: 0 | 0 |
| BadCamp (manual)<br>BadCamp (automatically) | $\{S, C\}$<br>None | Missing: 2<br>Additional: 0 | 0 |
| RDA-DMP (manual)<br>RDA-DMP (automatic) | $\{S, PF, C, R\}$<br>$\{S, C\}$ | Missed: 2<br>Additional: 0 | 1 |
| ArchiveSpace (manual)<br>ArchiveSpace (automatic) | $\{C, S, R, M\}$<br>$\{C, S\}$ | Missed: 2<br>Additional: 0 | 0 |
| UniBath (manual)<br>UniBath (automatic) | $\{C, S, R, M, PF\}$<br>$\{C, S\}$ | missed: 3<br>Additional: 0 | 0 |
| DuraSpace (manual)<br>DuraSpace (automatic) | $\{S\}$<br>$\{S\}$ | Missed: 0<br>Additional: 0 | 0 |
| RacDam (manual)<br>RacDam (automatic) | $\{S, C\}$<br>$\{S, C\}$ | Missed: 0<br>Additional: 0 | 0 |
| CulRepo (manual)<br>CulRepo (automatic) | $\{C, S, R, PF, PT\}$<br>$\{C, S\}$ | Missed: 3<br>Additional: 0 | 0 |
| Zooniverse (manual)<br>Zooniverse (automatic) | $\{C, S\}$<br>$\{C\}$ | Missed: 1<br>Additional: 0 | 0 |

## 4.5 Modèles plus simples et autres encodages

Dans cette section, nous avons voulu explorer des techniques de classification ML plus simples que l'apprentissage profond et les tester sur différents encodages pour notre tâche de détection de QA. Nous voulons voir si l'apprentissage profond avec SpaCy était une meilleure option. Nous avons donc effectué les mêmes expérimentations que pour SpaCy. Les différents modèles que nous allons tester sont LinearSVC[5], LogisticRegression[6] et ComplementNB[7] en utilisant l'implémentation de la librairie Sklearn car très simple d'utilisation. Ces différents modèles ne permettant pas une classification multi-label, nous devons donc utiliser la technique du One-Vs-Rest [Vapnik, 1995]. Pour chacun des classifieurs, nous allons les expérimenter avec l'encodage TF-IDF de Sklearn[8], Glove de Spacy [9] et BERT de la librairie Flair[10].

Chacune des expérimentations a été effectuée avec les données utilisées pour l'expérience similaire sur les modèles Spacy ; cela évitera un biais dans les résultats. Nous évaluons nos modèles avec les mêmes métriques que pour SpaCy (*Précision*, *Rappel* et *F1-Score*) avec une *10-fold cross-validation*.

### 4.5.1 Identification des US faisant référence à un attribut de qualité

Comme pour le modèle SpaCy, nous avons entraîné nos différentes options pour identifier si une US est liée ou non à un QA. A la table 4.9, nous pouvons retrouver les performances obtenues pour chaque modèle en fonction de l'encodage utilisé. Malheureusement, le modèle ComplementNB n'est pas compatible avec les encodages utilisant des valeurs négatives. Il est donc uniquement possible de l'évaluer sur l'encodage TF-IDF.

Table 4.9: Comparaison des résultats obtenus pour la classification QA/noQA par encodage pour chaque modèle avec une *10-fold cross-validation*

| | **TF-IDF** | | | **Glove** | | | **BERT** | | |
|---|---|---|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| LogisticReg | 0.71 | 0.67 | 0.69 | 0.63 | 0.64 | 0.63 | 0.7 | 0.7 | 0.7 |
| ComplementNB | 0.7 | 0.72 | 0.71 | - | - | - | - | - | - |
| LinearSVC | 0.69 | 0.68 | 0.69 | 0.61 | 0.71 | 0.65 | 0.68 | 0.69 | 0.68 |

Nous pouvons constater que c'est le modèle ComplementNB avec l'encodage TF-IDF qui obtient les meilleurs résultats avec un *F1-Score* de 0.71, une *Précision* de 0.7 et un *Rappel* de 0.72.

Comme pour le point 4.4.1, en nous inspirant de la pipeline proposée par M. Bhat et son équipe [Bhat et al., 2017], nous allons tester l'efficacité de nos différents modèles et encodages sur base d'US contenant au moins un QA.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[9] https://spacy.io/usage/vectors-similarity
[10] https://github.com/zalandoresearch/flair

**LogisticRegression**

A la Table 4.10, nous pouvons retrouver les résultats obtenus par le modèle LogisticRegression par QA en fonction des différents encodages. Cette fois encore, nous pouvons observer que c'est l'encodage de BERT qui permet d'obtenir les meilleurs résultats pour ce modèle avec un *F1-Score* de 0.51, une *Précision* de 0.59 et un *Rappel* de 0.47 pour *Global*.

Table 4.10: Comparaison des résultats pour l'identification du type de QA obtenus par encodage pour une LogisticRegression avec une *10-fold cross-validation* uniquement sur des US liées à au moins un QA

| | **TF-IDF** | | | **Glove** | | | **BERT** | | |
|---|---|---|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.3 | 0.0 | 0.3 | 0.42 | 0.22 | 0.41 | 0.45 | 0.08 | 0.39 |
| *compatibility* | 0.75 | 0.16 | 0.25 | 0.45 | 0.41 | 0.37 | 0.63 | 0.65 | 0.59 |
| *usability* | 0.0 | 0.0 | 0.0 | 0.22 | 0.12 | 0.12 | 0.57 | 0.38 | 0.43 |
| *reliability* | 0.2 | 0.0 | 0.2 | 0.32 | 0.09 | 0.22 | 0.4 | 0.08 | 0.32 |
| *security* | 0.6 | 0.17 | 0.22 | 0.36 | 0.25 | 0.28 | 0.72 | 0.75 | 0.66 |
| *maintainability* | 0.1 | 0.0 | 0.1 | 0.27 | 0.17 | 0.22 | 0.5 | 0.2 | 0.36 |
| *portability* | 0.4 | 0.0 | 0.4 | 0.3 | 0.0 | 0.3 | 0.36 | 0.03 | 0.34 |
| ***Global*** | **0.44** | **0.09** | **0.19** | **0.35** | **0.25** | **0.28** | **0.59** | **0.47** | **0.51** |

**ComplementNB**

A la Table 4.11, nous pouvons retrouver les résultats obtenus par le modèle LogisticRegression par QA. Les limitations de ce modèle l'empêchent d'utiliser des encodages générant des valeurs négatives. Nous ne pourrons donc l'évaluer que sur l'encodage TF-IDF.

Table 4.11: Résultats obtenus pour l'identification du type de QA pour une ComplementNB avec l'encodage TF-IDF avec une *10-fold cross-validation* uniquement sur des US liées à au moins un QA

| | **TF-IDF** | | |
|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.21 | 0.2 | 0.22 |
| *compatibility* | 0.58 | 0.64 | 0.57 |
| *usability* | 0.63 | 0.43 | 0.49 |
| *reliability* | 0.03 | 0.05 | 0.04 |
| *security* | 0.66 | 0.78 | 0.67 |
| *maintainability* | 0.24 | 0.23 | 0.2 |
| *portability* | 0.41 | 0.16 | 0.38 |
| ***Global*** | **0.5** | **0.5** | **0.47** |

**LinearSVC**

A la Table 4.12, nous pouvons retrouver les résultats obtenus par le modèle LinearSVC par QA en fonction des différents encodages. Cette fois encore, nous pouvons constater que l'encodage BERT est le plus performant au niveau *Global* avec un *F1-Score* de 0.48 , une *Précision* de 0.52 et un *Rappel* de 0.51.

Table 4.12: Comparaison des résultats obtenus pour l'identification du type de QA par encodage pour une LinearSVC avec une *10-fold cross-validation* uniquement sur des US liées à au moins un QA

| | **TF-IDF** | | | **Glove** | | | **BERT** | | |
|---|---|---|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.4 | 0.05 | 0.37 | 0.27 | 0.32 | 0.31 | 0.35 | 0.08 | 0.29 |
| *compatibility* | 0.58 | 0.43 | 0.48 | 0.44 | 0.41 | 0.35 | 0.63 | 0.69 | 0.61 |
| *usability* | 0.68 | 0.2 | 0.3 | 0.15 | 0.16 | 0.12 | 0.54 | 0.43 | 0.45 |
| *reliability* | 0.4 | 0.05 | 0.28 | 0.26 | 0.18 | 0.25 | 0.1 | 0.08 | 0.09 |
| *security* | 0.84 | 0.63 | 0.67 | 0.31 | 0.25 | 0.26 | 0.67 | 0.78 | 0.66 |
| *maintainability* | 0.37 | 0.08 | 0.22 | 0.15 | 0.19 | 0.15 | 0.34 | 0.27 | 0.24 |
| *portability* | 0.6 | 0.04 | 0.47 | 0.0 | 0.0 | 0.0 | 0.32 | 0.09 | 0.3 |
| ***Global*** | **0.61** | **0.33** | **0.44** | **0.29** | **0.27** | **0.24** | **0.52** | **0.51** | **0.48** |

**Comparaison**

En comparant l'ensemble des résultats précédents, nous pouvons constater que c'est le modèle LogisticRegression avec l'encodage BERT qui obtient les meilleurs résultats avec un *F1-Score* de 0.51, une *Précision* de 0.59 et un *Rappel* de 0.59 pour *Global*. Nous pouvons constater qu'il va obtenir de bons résultats pour la *security* et la *compatibility* avec un *F1-Score* de 0.66 et 0.59. Nous avons la *usability* et la *performance* qui restent acceptables avec un *F1-Score* proche de 0.4. Pour les autres QA, leurs résultats deviennent moins bons avec un *F1-Score* inférieur ou égal à 0.36.

Si nous analysons les trois QAs les mieux prédits pour chaque modèle, nous pouvons constater que *compatibility* et *security* (à l'exception de LogisticRegression avec Doc2Vec) en font toujours partie. On peut aussi constater que *usability*, *reliability* et *maintainability* composent très souvent le trio des moins bien identifiés.

### 4.5.2 Identification des attributs de qualité liés à une US

Comme pour le modèle SpaCy, nous avons évalué nos différents modèles sur leur capacité à identifier les QAs liés à une US à l'aide d'une *10-fold cross-validation*. Nous utiliserons les mêmes métriques d'évaluation (*Précision*, *Rappel* et *F1-Score*). Les données utilisées seront les mêmes que pour l'entraînement du modèle SpaCy au point 4.4.3.

**LogisticRegression**

A la Table 4.13, nous pouvons retrouver les résultats obtenus par le modèle LogisticRegression par QA en fonction des différents encodages. Nous pouvons observer que c'est l'encodage de BERT qui permet d'obtenir les meilleurs résultats pour ce modèle.

Table 4.13: Comparaison des résultats obtenus pour l'identification du type de QA par encodage pour une LogisticRegression avec une *10-fold cross-validation*

| | **TF-IDF** | | | **Glove** | | | **BERT** | | |
|---|---|---|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.3 | 0.17 | 0.2 | 0.58 | 0.36 | 0.41 | 0.67 | 0.42 | 0.49 |
| *compatibility* | 0.9 | 0.21 | 0.33 | 0.5 | 0.28 | 0.35 | 0.7 | 0.55 | 0.6 |
| *usability* | 0.0 | 0.0 | 0.0 | 0.33 | 0.05 | 0.09 | 0.46 | 0.3 | 0.34 |
| *reliability* | 0.1 | 0.0 | 0.1 | 0.23 | 0.05 | 0.16 | 0.45 | 0.11 | 0.26 |
| *security* | 0.2 | 0.02 | 0.03 | 0.4 | 0.11 | 0.16 | 0.63 | 0.54 | 0.57 |
| *maintainability* | 0.25 | 0.09 | 0.12 | 0.36 | 0.21 | 0.25 | 0.47 | 0.35 | 0.39 |
| *portability* | 0.0 | 0.0 | 0.0 | 0.2 | 0.07 | 0.1 | 0.43 | 0.26 | 0.31 |
| ***Global*** | **0.4** | **0.1** | **0.15** | **0.41** | **0.18** | **0.24** | **0.59** | **0.43** | **0.48** |

**ComplementNB**

A la table 4.14, nous pouvons retrouver les résultats obtenus pour le modèle ComplementNB par QA uniquement avec l'encodage TF-IDF.

Table 4.14: Résultats obtenus pour l'identification du type de QA pour une ComplementNB avec l'encodage TF-IDF avec une *10-fold cross-validation*

| | **TF-IDF** | | |
|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.39 | 0.4 | 0.36 |
| *compatibility* | 0.68 | 0.55 | 0.6 |
| *usability* | 0.37 | 0.28 | 0.3 |
| *reliability* | 0.27 | 0.11 | 0.13 |
| *security* | 0.59 | 0.48 | 0.51 |
| *maintainability* | 0.51 | 0.56 | 0.52 |
| *portability* | 0.33 | 0.41 | 0.35 |
| ***Global*** | **0.53** | **0.45** | **0.47** |

**LinearSVC**

A la table 4.15, nous pouvons retrouver les résultats obtenus par le modèle LinearSVC par QA en fonction des différents encodages. Cette fois encore, nous pouvons observer que c'est l'encodage de BERT qui permet d'obtenir les meilleurs résultats pour ce modèle.

Table 4.15: Comparaison des résultats obtenus pour l'identification du type de QA par encodage pour une LinearSVC avec une *10-fold cross-validation*

| | TF-IDF | | | Glove | | | BERT | | |
|---|---|---|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.65 | 0.35 | 0.44 | 0.46 | 0.54 | 0.46 | 0.7 | 0.55 | 0.59 |
| *compatibility* | 0.79 | 0.47 | 0.58 | 0.49 | 0.3 | 0.34 | 0.61 | 0.61 | 0.59 |
| *usability* | 0.45 | 0.11 | 0.17 | 0.14 | 0.06 | 0.06 | 0.38 | 0.31 | 0.32 |
| *reliability* | 0.5 | 0.15 | 0.3 | 0.11 | 0.18 | 0.12 | 0.33 | 0.16 | 0.28 |
| *security* | 0.83 | 0.42 | 0.55 | 0.28 | 0.11 | 0.15 | 0.58 | 0.61 | 0.59 |
| *maintainability* | 0.72 | 0.43 | 0.5 | 0.3 | 0.27 | 0.27 | 0.43 | 0.39 | 0.39 |
| *portability* | 0.3 | 0.2 | 0.21 | 0.12 | 0.17 | 0.14 | 0.33 | 0.28 | 0.28 |
| ***Global*** | **0.68** | **0.36** | **0.45** | **0.33** | **0.22** | **0.23** | **0.52** | **0.49** | **0.49** |

**Comparaison**

Nous pouvons observer que c'est le LinearSVC avec l'encodage BERT qui obtient les meilleures performances *Global* avec une *Précision* de 0.52, un *Rappel* de 0,49 et un *F1-Score* de 0.49. Nous observons que l'éventail des *F1-Score* par QA est très large. Il va de 0.28 pour le plus mauvais jusqu'à 0.59 pour le meilleur. On peut donc constater que *compatibility*, *performance* et *security* sont classifiés de façon correcte à un niveau acceptable avec un *F1-Score* proche de 0.6. Pour *maintainability*, la classification est plus médiocre, mais pas totalement inacceptable avec un *F1-Score* aux alentours de 0.4. Finalement, nous constatons que notre classifieur est extrêmement mauvais pour identifier *portability*, *usability* et *reliability* avec un *F1-Score* proche de 0.3.

En observant l'ensemble des résultats précédents, nous pouvons constater que les trois premières places du podium des QAs qui sont les mieux prédits, se jouent entre *performance*, *compatibility*, *maintainability* et *security*. Pour les moins bien prédits, cela se jouera le plus souvent entre *usability*, *portability* et *reliability*.

## 4.6 Discussion

### 4.6.1 Variation des évaluations par attribut de qualité

Comme nous avons pu l'observer à la Table 4.5 , les résultats obtenus pour chaque QA sont très variés, de corrects à très mauvais. Pour expliquer cette forte différence, nous avons mis en lien à la Table 4.16 le nombre d'occurrences dans notre base de données pour chaque QA avec leur score. Nous avons pu en déduire que le nombre d'exemples pour chaque QA a une importance majeure pour les résultats car plus un QA en possède, plus ses résultats sont élevés. On peut voir que les QAs possédant des résultats acceptables sont ceux avec le plus d'occurrences. On peut constater la même chose pour les autres résultats avec un nombre moins élevé d'occurrences. Toutefois, nous pouvons remarquer un contre-exemple, *usability* a le troisième plus grand nombre d'exemples mais il possède des résultats très médiocres. Nous expliquons cela par la nature même de ce QA.

L'*usability* correspond à un ensemble très varié de domaines et ne possède pas de vocabulaire

Table 4.16: Comparaison du nombre d'occurrences d'un QA par rapport à son *F1-Score* obtenu par une *10-fold cross-validation*

| QA | Nombre | *F1-Score* |
|---|---|---|
| *compatibility* | 165 | 0.63 |
| *security* | 97 | 0.57 |
| *usability* | 80 | 0.29 |
| *maintainability* | 57 | 0.59 |
| *performance* | 31 | 0.39 |
| *portability* | 25 | 0.44 |
| *reliability* | 28 | 0.18 |

spécifique. Dans la Figure 4.5 modélisant le nombre d'occurrences des 10 mots les plus utilisés dans les US liées à *usability*, nous pouvons constater qu'aucun mot n'est directement lié à ce QA et le nombre d'occurrences moyen (10 par mot en retirant User qui est du bruit) reste très faible proportionnellement au nombre d'US. Etant donné que *usability* ne concerne pas un sujet précis, il est très compliqué pour un modèle de la prédire avec efficacité. Nous pensons qu'il pourrait être judicieux de trouver des sous-catégories moins vagues à l'intérieur de *usability* qui seront plus aisées à prédire.

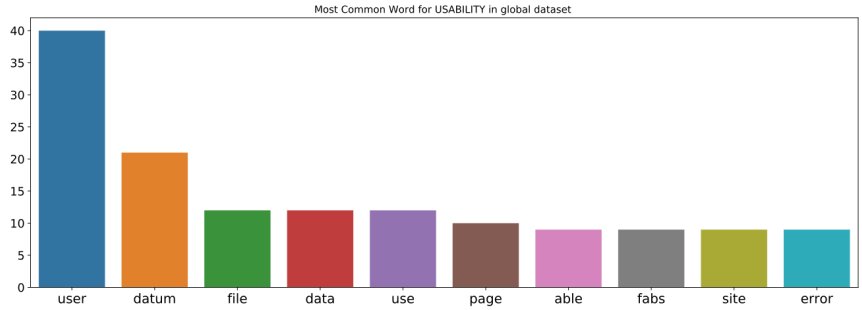


Figure 4.5: Mots les plus fréquents dans les US liées à *usability*

Pour venir renforcer nos hypothèses, nous pouvons utiliser les observations du point 4.5.1. Celles-ci nous permettent de remarquer une corrélation entre l'ordre général des performances par QA. Pour *Rappel*, nous avions constaté que les premières places se jouaient entre *performance*, *compatibility*, *maintainability* et *security* et pour les dernières entre *usability*, *portability* et *reliability*. Nous pouvons constater qu'une fois encore les plus performantes correspondent à celles les plus représentées dans la base de données et inversement pour celles les moins représentées. Cette fois encore, *usability* obtient des résultats médiocres malgré son nombre d'occurrences assez important, ce qui vient appuyer notre hypothèse.

### 4.6.2 Équilibrage de la base de données

Notre base de données complète est très mal équilibrée. Pour le QA le plus référencé (*compatibility*), nous avons seulement 10% des US liées à celui-ci et pour celui qui l'est le moins (*portability*), nous en avons moins de 2%. Ce fort déséquilibre poussera nos modèles à ne jamais prédire la présence d'un QA dans une US. Nous avons donc décidé arbitrairement de n'utiliser que 33% des US ne faisant référence à aucun QA pour évaluer nos modèles dans la tâche d'identification du type de QA auquel une US fait référence. Ces 33% d'US ont été choisis aléatoirement parmi toutes celles ne faisant référence à aucun QA. Nous avons choisi 33% car nous avons voulu trouver un juste milieu entre utiliser l'ensemble des US qui ne font pas référence à un QA et n'en n'utiliser aucune.

À la Table 4.17, nous comparons les résultats obtenus avec SpaCy sur une base de données possédant 33% des US qui ne sont pas liées à un QA et ceux obtenus avec uniquement des US liées à au moins un QA. Nous pouvons observer que plus le nombre d'US sans QA est faible, plus le *Rappel* augmente (0.41 à 0.53) et à l'inverse, la *Précision* diminue (0.74 à 0.71). Nous pouvons expliquer ces observations par le fait que moins nous avons d'US sans QA, plus la base de données est équilibrée par rapport à un QA. Notre modèle prédit donc plus souvent ce QA.

Table 4.17: Performances obtenues par un modèle SpaCy avec 33% d'US faisant référence à aucun QA et 0% d'US faisant référence à aucun QA

| | **33% d'US sans QA** | | | **0% d'US sans QA** | | |
|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.57 | 0.31 | 0.39 | 0.62 | 0.42 | 0.47 |
| *compatibility* | 0.77 | 0.55 | 0.63 | 0.74 | 0.69 | 0.71 |
| *usability* | 0.61 | 0.20 | 0.29 | 0.66 | 0.46 | 0.52 |
| *reliability* | 0.29 | 0.14 | 0.18 | 0.2 | 0.08 | 0.11 |
| *security* | 0.66 | 0.50 | 0.57 | 0.77 | 0.64 | 0.69 |
| *maintainability* | 0.77 | 0.50 | 0.59 | 0.69 | 0.56 | 0.58 |
| *portability* | 0.56 | 0.39 | 0.44 | 0.6 | 0.38 | 0.45 |
| ***Global*** | **0.74** | **0.41** | **0.53** | **0.71** | **0.53** | **0.6** |

En conclusion, le nombre d'US sans QA utilisé influencera le nombre d'identifications réalisées par nos modèles, mais par conséquent le nombre de faux positifs augmentera lui aussi. Afin de choisir une valeur moins arbitraire, il pourrait être intéressant d'analyser plus en détail son influence afin de faire le meilleur choix selon les besoins tout en obtenant un nombre de faux positifs acceptable et en prédisant un maximum de QA.

### 4.6.3 Seuil de décision

Selon les observations effectuées pour différents seuils de décision à la Table 4.6, nous pouvons constater que la modification d'un seuil a une influence sur nos résultats. Au vu de nos petits jeux de données, le gain d'une prédiction juste est très important à notre échelle. La seconde observation est que les effets d'un changement de seuil varient selon le QA. Nous pensons donc que dans des travaux futurs qui auront pour but d'augmenter les performances de nos modèles, nous pourrions

chercher à optimiser le seuil en fonction de chaque QA pour améliorer ses résultats. Toutefois, il sera nécessaire de veiller à ne pas créer des problèmes d'overfitting *i.e.* surapprentissage sur les données au détriment de la généralisation des prédictions. Dans la comparaison effectuée à la Table 4.6, nous avons pu constater que le choix d'un différent seuil selon le QA a une influence sur ses résultats. Nous pensons donc que l'optimisation du choix du seuil en fonction du QA est une piste pour des travaux futurs.

### 4.6.4 Comparaison SpaCy par rapport aux autres modèles testés

Pour savoir si l'apprentissage profond avec SpaCy était une meilleure option que d'autres types de modèles de classification pour nos deux tâches, nous comparons les résultats obtenus pour la tâche de détection des US faisant référence à un QA ou non. Nous pouvons constater que le modèle ComplementNB avec TF-IDF a des résultats significativement meilleurs que ceux obtenus avec SpaCy. Il a une amélioration de 0.05 pour le *F1-Score* (avec 0.71), de 0.05 pour la *Précision* (avec 0.7) et de 0.05 pour le *Rappel* (avec 0.72) par rapport à SpaCy.

Pour ce qui est de l'entraînement d'un modèle capable d'identifier le type de QA sur des US faisant uniquement référence à des QAs inspirés des travaux de Bhat et son équipe [Bhat et al., 2017], nous pouvons retrouver à la table 4.18, la comparaison des performances de SpaCy avec celles obtenues par LogisticRegression avec BERT (a obtenu les meilleurs résultats parmi les autres modèles). Nous pouvons constater que c'est SpaCy qui domine largement cette tâche avec des résultats significativement meilleurs.

Table 4.18: Comparaison des performances obtenues entre SpaCy et LogisticRegression utilisant l'encodage BERT uniquement sur des US liées au moins à un QA

| | **SpaCy** | | | **LogReg + BERT** | | |
|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.62 | 0.42 | 0.47 | 0.45 | 0.08 | 0.39 |
| *compatibility* | 0.74 | 0.69 | 0.71 | 0.63 | 0.65 | 0.59 |
| *usability* | 0.66 | 0.46 | 0.52 | 0.57 | 0.38 | 0.43 |
| *reliability* | 0.2 | 0.08 | 0.11 | 0.4 | 0.08 | 0.32 |
| *security* | 0.77 | 0.64 | 0.69 | 0.72 | 0.75 | 0.66 |
| *maintainability* | 0.69 | 0.56 | 0.58 | 0.5 | 0.2 | 0.36 |
| *portability* | 0.6 | 0.38 | 0.45 | 0.36 | 0.03 | 0.34 |
| ***Global*** | **0.71** | **0.53** | **0.6** | **0.59** | **0.47** | **0.51** |

En résumé, nous pouvons constater que notre intuition de départ n'était pas totalement bonne. L'apprentissage profond a l'air effectivement meilleur pour la tâche complexe d'identification des types de QA présents dans une US mais ne l'est pas pour celle de détection des US faisant référence à au moins un QA.

Pour ce qui est de l'idée de découper notre pipeline en deux étapes, la première identifiant les US faisant référencent à au moins un QA et la seconde identifiant le type de celles-ci, nous pensons que les gains de performance obtenus en entraînant/évaluant nos modèles uniquement sur US

liées à au moins un QA ne sont pas suffisants au vu d'un nombre d'US perdues lors de la première étape. Ce processus en deux étapes peut se justifier si nous désirons diminuer le nombre de faux positifs en réduisant le nombre d'US à seulement celles identifiées comme faisant référence à au moins un QA.

Ensuite, nous avons comparé à la Table 4.19 les résultats obtenus avec SpaCy par rapport aux meilleurs observés pour les autres approches pour la tâche d'identification des QAs par US sur une base de données composée d'US avec QA ou non. Comme nous avons pu le constater précédemment, c'est le modèle LinearSVC avec l'encodage BERT qui a obtenu les meilleurs résultats parmi les différents modèles et encodages testés. C'est donc lui qui sera représenté dans la Table 4.19.

Table 4.19: Comparaison des performances obtenues entre SpaCy et LinearSVC utilisant l'encodage BERT

| | **SpaCy** | | | **LinearSVC + BERT** | | |
|---|---|---|---|---|---|---|
| **QA** | $prec$ | $rec$ | $f_1$ | $prec$ | $rec$ | $f_1$ |
| *performance* | 0.57 | 0.31 | 0.39 | 0.7 | 0.55 | 0.59 |
| *compatibility* | 0.77 | 0.55 | 0.63 | 0.61 | 0.61 | 0.59 |
| *usability* | 0.61 | 0.20 | 0.29 | 0.38 | 0.31 | 0.32 |
| *reliability* | 0.29 | 0.14 | 0.18 | 0.33 | 0.16 | 0.28 |
| *security* | 0.66 | 0.50 | 0.57 | 0.58 | 0.61 | 0.59 |
| *maintainability* | 0.77 | 0.50 | 0.59 | 0.43 | 0.39 | 0.39 |
| *portability* | 0.56 | 0.39 | 0.44 | 0.33 | 0.28 | 0.28 |
| ***Global*** | **0.74** | **0.41** | **0.53** | **0.52** | **0.49** | **0.49** |

Pour cette troisième comparaison, SpaCy confirme sa supériorité avec un *F1-Score* 4% supérieur à son concurrent pour *Global*. Toutefois, il est intéressant de faire remarquer que malgré ses performances moindres, LinearSVC obtient en moyenne un *F1-Score* largement supérieur à celui de Spacy pour la *performance*. Nous avons pensé de cette observation qu'une approche One-VS-Rest composée de différents modèles pourrait permettre de combler les faiblesses d'autres modèles vis-à-vis de certains QA. Par exemple, nous pourrions utiliser le modèle SpaCy pour tous les QA, à l'exception de *performance* qui serait prédite par le LinearSVC.

### 4.6.5 Une meilleure option : BERT

Par manque de temps, nous n'avons pas encore pu explorer la possibilité d'utiliser une pipeline dans laquelle BERT joue le rôle d'encodeur, mais aussi celui de classifieur. Toutefois, si nous comparons les résultats obtenus à l'aide de l'encodage proposé par BERT avec ceux de Glove (SpaCy), nous pouvons observer qu'en général BERT permet un gain important de performances. Ces analyses nous poussent donc à penser que l'utilisation de BERT (encodage + classification) pourrait permettre un gain non négligeable de performances pour nos différentes tâches.

### 4.6.6 Evaluation par backlog

La vue d'ensemble des QA et de leur importance par nombre d'occurrences que nous proposons au Point 4.4.6 (Evaluation par backlog) possède plusieurs limitations. Premièrement, elle est limitée par le fait que nous considérons tous les QAs sur un pied d'égalité. Une analyse plus précise d'un QA pourrait montrer une importance plus grande malgré son faible nombre d'occurrences. Deuxièmement, la valeur business des US n'est pas prise en compte dans notre classement. Il serait intéressant d'ajouter une notion de poids qui permettrait d'augmenter la priorité d'un QA en fonction de la valeur business des US auxquelles il est lié. Troisièmement, nous ne prenons pas en compte les changements de taille du backlog à travers la vie du projet. Toutefois, notre approche permet une vue d'ensemble du projet à une étape de son déroulement, ce qui reste une information précieuse pour permettre aux développeurs de mieux appréhender leurs prochaines décisions de design.

## 4.7 Conclusion

Les différents attributs de qualité (QA), en fonction de leur importance dans un projet, influencent grandement les décisions de design. Il est donc crucial de les identifier rapidement. Pour cela, nous proposons une solution automatique de détection des histoires d'utilisateur (US) faisant référence à un QA et d'identification du type de QA à l'aide du ML.

Pour valider notre idée, nous avons comparé les résultats obtenus par un ensemble de modèles de classification d'apprentissage automatisé (SpaCy[11], LinearSVC[12], LogisticRegression[13] et ComplementNB[14]) avec différents encodages (TF-IDF, Glove et BERT) dans ces deux tâches. C'est le modèle ComplementNB avec l'encodage TF-IDF qui l'emporte avec un *F1-Score Global* de 0.71 pour la détection des US faisant référence à un QA et c'est SpaCy qui obtient le meilleur *F1-Score Global* pour l'identification du type de QA présent dans une US avec 0.53. Nous avons pu montrer que l'augmentation de la taille de notre base de données et l'optimisation du seuil de décision en fonction de l'attribut de qualité sont des facteurs clés pour l'amélioration de nos résultats. Pour mesurer l'importance d'un QA pour un projet, nous avons proposé un classement généré sur base du nombre d'US identifiées pour chaque QA pour le backlog d'un projet. Celui-ci n'est malheureusement pas assez pertinent car il ne prend pas en compte la valeur business de chaque histoire d'utilisateur.

[11] https://spacy.io/
[12] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[13] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[14] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html

CHAPTER 5

# LEGACY

*Dans ce chapitre, nous allons présenter l'objectif du projet "From User Stories to Use Case Scenarios - Towards a Generative Approach" [Gilson and Irwin, 2018] et son état d'avancement avant de travailler sur celui-ci. Pour éviter toute confusion entre cet état du projet et nos futures améliorations, nous le nommerons Legacy. Dans la première section, nous aborderons les motivations et les objectifs de ce projet de recherche. Dans la seconde section, nous décrirons le projet déjà existant en expliquant les différents modules qui le composent, les limitations et les divers modifications techniques que nous avons dû lui faire subir.*

## 5.1 Introduction

### 5.1.1 Motivations et objectifs

F. Gilson et M. Galster. ont démarré la conception d'un logiciel d'assistance au développement qui proposera une multitude de fonctionnalités permettant de mieux conserver la connaissance d'un projet et d'aider les développeurs dans leurs choix architecturaux. La première étape se base sur l'analyse automatique du backlog et l'extraction d'informations depuis celui-ci. Pour ce faire, ils se sont tournés vers différents outils NLP et techniques du ML. Comme expliqué dans le Chapitre 2, l'utilisation d'US est devenue très populaire en méthode Agile [Schön et al., 2017] mais l'augmentation de la taille du Backlog rend la tâche d'analyse et de maintien par les architectes logiciels de plus en plus difficile[Bass, 2016]. Pour la rendre plus aisée, ils proposent une modélisation automatique des US sous forme de diagrammes de robustesse [Jacobson, 1987] qui permettront une visualisation simplifiée des liens entre les différents éléments qui composent le projet.

Ils ont choisi d'utiliser le diagramme de robustesse pour modéliser les US car celui-ci permet

de visualiser simplement et clairement le flux d'action décrit dans des US. Ils ont pu appuyer leur décision sur d'autres recherches ayant prouvé l'utilité de ce type de diagramme dans l'industrie [Rosenberg and Stephens, 2007; El-Attar and Miller, 2010]. Dans sa forme actuelle , le diagramme de robustesse ne permet pas de représenter clairement les sous-composants du système. Ils ont donc pris la décision d'introduire le type d'élément *Property* permettant de modéliser ces sous-éléments et ceux-ci seront liés à leur *Entity* parent par une relation de composition (-<>). De plus, ils ont ajouté une relation permettant de modéliser la dépendance entre deux *Entity* (- ->).

Pour mieux comprendre comment transformer des US en diagramme de robustesse, prenons un exemple. Pour celui-ci, nous allons utiliser l'US :

*"As an Applicant, I want to check the status of a transaction"*

D'abord, nous allons identifier l'*Actor* qui sera toujours cité dans la *part-<acteur>*. Ici, ce sera *"Applicant"*. Ensuite, pour trouver l'ensemble des autres éléments, nous devrons analyser la *part-<objectif>* de l'US ; la *part-<bénéfice>* n'est pas utilisée pour la modélisation. Le *Control* correspond au verbe cité après le *"want"*. Dans ce cas, c'est *"check"*. Nous pouvons retrouver à sa suite la *Property "status"* venant composer l'*Entity "transaction"* sur laquelle il interagit. On peut constater qu'il n'est pas mentionné d'interface. Très peu d'US y font référence et il a été décidé de les induire pour générer des diagrammes syntaxiquement corrects. Nous pouvons proposer *"Transaction Interface"* qui est un bon candidat. Une fois que tous les éléments clés composant notre US sont identifiés, nous pouvons ajouter les différentes relations entre chacun d'eux. À la figure 5.1, nous pouvons retrouver le mapping entre l'arbre des dépendances de notre exemple et les éléments de son DR.

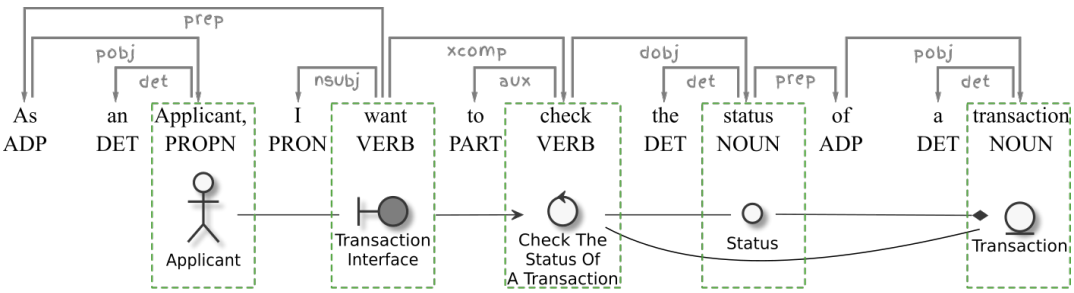


Figure 5.1: Exemple de mapping entre les mots d'une US et les objets de son DR [Gilson et al., 2019b]

La force de cette modélisation c'est qu'elle rend beaucoup plus aisée l'identification des éléments à ajouter au système, les interfaces nécessaires et les différentes interactions entre chacun. L'utilité de cette modélisation sur une seule US est assez limitée, mais si nous prenons un ensemble d'US et que nous fusionnons leur visualisation pour obtenir un DR unifié, il sera aisé d'identifier :

- les *Boundary* clés par lesquelles les *Actor* affectent le système;
- l'ensemble des *Entity* composant le domaine du projet et les *Control* interagissant avec elles;
- les éléments communs entre plusieurs US.

L'unification des éléments partagés par plusieurs US permettra d'identifier rapidement les dépendances et les interactions entre les éléments composant chaque US. Cette visualisation d'un backlog aura comme but d'aider à avoir une meilleure vue d'ensemble du projet afin d'aider dans la prise de décisions de design architectural.

### 5.1.2 Méthode de recherche

Les développeurs de Legacy se sont inspirés des recherches présentées précédement qui utilisent l'arbre des dépendances d'une phrase pour la convertir en un diagramme [Osman and Zalhan, 2016; Friedrich et al., 2011; Treude et al., 2015]. Pour pouvoir vérifier leurs hypothèses selon lesquelles l'utilisation de l'arbre et d'autres outils NLP peut permettre la modélisation automatique d'une US en DR, ils ont défini eux-mêmes un ensemble d'exemples d'US simples (+- 30).

Ils ont effectué une recherche des différents outils permettant de générer des DR. L'outil qui leur a paru le plus intéressant est la librairie OpenSource plantUML[1]. Elle a été désignée comme le meilleur choix grâce à sa simplicité d'utilisation et sa syntaxe lisible par un humain, contrairement à d'autres options comme XML Metadata Interchange.

## 5.2 Implémentation du Legacy

Dans cette section, nous allons d'abord décrire chacun des objets clés qui seront utilisés au travers de cette implémentation. Ensuite, nous expliquerons la Pipeline de Legacy. Nous aborderons par après les différentes limitations du projet à ce stade. Finalement, nous expliquerons chacune des modifications techniques que nous avons effectuées sur les concepts clés.

### 5.2.1 Objets clefs

#### DiagramObject

DiagramObject (DO) correspond aux différents éléments qui composeront le diagramme de robustesse. Chacun d'entre eux sera composé :

- **pretty-name** : la description utilisée pour le diagramme de robustesse;
- **children** : la liste des DO qui sont ses enfants;
- **parents** : la liste des DO qui sont ses parents;
- **object-type** : son type (*Entity*, *Actor*, *Control*, *Boundary* ou *Property*);
- **explicit** : si l'objet a été découvert de façon explicite;
- **root** : mot principal *e.g.*, le verbe représentant l'action pour un *Control*;
- **specialisation** : identifiant créé sur base du type, du root et de ses différentes relations, si nécessaire.

[1] http://plantuml.com/fr/

#### Entité parsée

Chaque entité parsée correspond à un mot de l'US identifiée comme un potentiel candidat pour la modélisation finale. Chaque entité est donc composée :

- **Noyau** : mot principal;
- **Lemma** : lemmatisation du noyau;
- **Type d'entité** : type auquel l'entité a été identifiée par le parseur;
- **Propriété** : propriété qui est liée à cette entité;
- **Relations** : ensemble de ses entités enfants;
- **Children** : ensemble des entités qui la composent (uniquement quand l'entité est un composite);
- **Référence explicite** : Vrai si l'entité est explicite;
- **Description, case et pretext** : sont des informations supplémentaires utiles pour la description d'un objet du diagramme.

Il existe une variance des entités qui sont des Composites. Ceux-ci possèdent les mêmes attributs. Leur particularité est qu'ils sont composés de plusieurs entités reprises dans leurs Childrens. Celles-ci sont utilisées pour lier différents éléments comme par exemple pour les conjonctions.

Le type d'une entité est l'une des informations les plus importantes car c'est grâce à elle que l'on pourra déduire la place qu'occupera l'entité dans la modélisation finale. Ci-dessous, nous expliquons pour chaque type d'entité parsée comment elle sera représentée sur le diagramme de robustesse.

- **Acteur** : devient un *Actor*
- **Interface** : devient une *Boundary*
- **Relation** : devient un *Control*
- **Objet** : devient une *Entity*
- **Extension** : information venant compléter la description d'un autre objet du diagramme ; elle peut aussi devenir une *Property*.
- **Composite** : deviendra ce dont il est composé.
- **Actor-proxy, Primary-actor-proxy** : référence à l'*Actor* principal qui sera modélisé comme cet *Actor* si nécessaire.
- **Objects-proxy** : référence à une entité Objet qui sera modélisée comme cette entité Objet si nécessaire (donc en *Entity*).

Les propriétés comme les entités sont générées à partir des règles de parsing. Elles sont uniquement composées de leur noyau (mot principal) et de leur propriété parent et enfant si elles en ont. Celles-ci serviront à compléter la description des objets du diagramme ou pour modéliser des objets *Property* dans le diagramme.

#### *Carryover*

Le *Carryover* est l'outil utilisé pour garder en mémoire des informations sur l'état actuel du parsing. Par exemple, il permet de savoir si nous sommes dans le cas d'une conjonction. Il garde en mémoire

les différents sujets identifiés et les différentes références trouvées les concernant. Cette approche fonctionne pour les US simples mais elle sera limitée et causera parfois des erreurs de modélisation. A l'heure actuelle, il n'existe pas de technique très efficace pour le co-référencement donc celle-ci reste acceptable vu le peu d'erreurs observées.

#### Règles de parsing

Les règles de parsing sont les éléments clés permettant au parseur de savoir que faire pour chaque noeud de l'arbre des dépendances. Chacune possède une condition vérifiable par un seul type de noeud. Celle-ci permet de savoir si l'on doit appliquer ou non la règle sur le noeud actuel. Nous décrirons dans le prochain chapitre l'ensemble des règles définies pour ce projet car un grand nombre d'entre elles ont dû être modifiées. Toutefois, nous pouvons décrire ses inputs et outputs inchangés. Une règle prendra en entrée le noeud à parser qui comble sa condition, l'entité parsée Parent si elle existe et l'état actuel du parseur contenu dans le *Carryover*. En sortie, elle renverra l'entité parsée qui sera le prochain parent (très souvent celle-ci correspond à l'entité générée sur base du noeud), la liste des entités parsées à ajouter à la liste des candidats, le *Carryover* modifié et les prochains noeuds enfants à parser.

### 5.2.2 Pipeline

Pour proposer une modélisation automatique, les créateurs se sont basés sur les exemples présentés précédemment [Osman and Zalhan, 2016; Friedrich et al., 2011; Treude et al., 2015]. Ils vont donc opter pour l'utilisation de différents mécanismes NLP pour extraire les éléments du diagramme et leurs liens.

La première étape de leur pipeline est la génération d'arbre des dépendances et le POS grâce à la librairie SpaCy qui selon Al Omran et C. Treude [Al Omran and Treude, 2017] est un des meilleurs choix vu sa facilité d'utilisation et son efficacité.

La seconde étape consiste à sélectionner les branches de l'arbre correspondant à une des trois parties de l'US (*part-<acteur>*, *part-<objectif>*, *part-<bénéfice>*) et les parser pour identifier dans la première partie : les *Actor*, dans la seconde : les *Boundary*, *Control* et *Entity* et la troisième sera utilisée pour l'extraction de *Use Case* (ceux-ci ont été considérés comme superflus, ils ont donc été supprimés de la modélisation). Pour réaliser cette identification, ils ont défini un parseur composé d'un ensemble de règles basées sur les dépendances et le POS de chaque mot.

La dernière étape consistera en la génération du code plantUML permettant la modélisation du diagramme de robustesse final sur base des entités parsées. Tout d'abord, ils génèrent des DiagramObject (DO) à partir des entités parsées par US à l'exception des Extensions et des Proxys.

Ensuite, ils vont ajouter les différentes relations entre les DO générés dans l'étape précédente sur base des relations entre les entités parsées et transformer les propriétés de celles-ci en DO et les lier (pour éviter un trop grand nombre de *Property* modélisées seules, ils séparent la propriété seulement si l'objet auquel elle se rapporte est utilisé dans un autre cas). Comme nous l'avons expliqué précédemment, énormément d'US n'ont pas d'interface décrite. Ils vont donc en inférer

une pour chaque *Control* qui n'en a pas sur base des *Entity* liées à celui-ci et relier cette nouvelle *Boundary* au *Control* et à l'*Actor* qui le réalise.

Puis, ils vont fusionner l'ensemble des DO ayant une spécialisation commune pour unifier toutes les occurrences où une même entité est utilisée. Cette fusion consiste à créer un DO possédant tous les liens et informations des éléments partageant la même spécialisation. Cette étape permet d'unifier les différentes US partageant des DO.

Finalement, ils vont générer le code plantUML sur base des DO unifiés dans l'étape précédente et des relations qu'ils partagent entre eux. Le type de chaque relation sera défini en fonction du type des deux objets qui la composent.

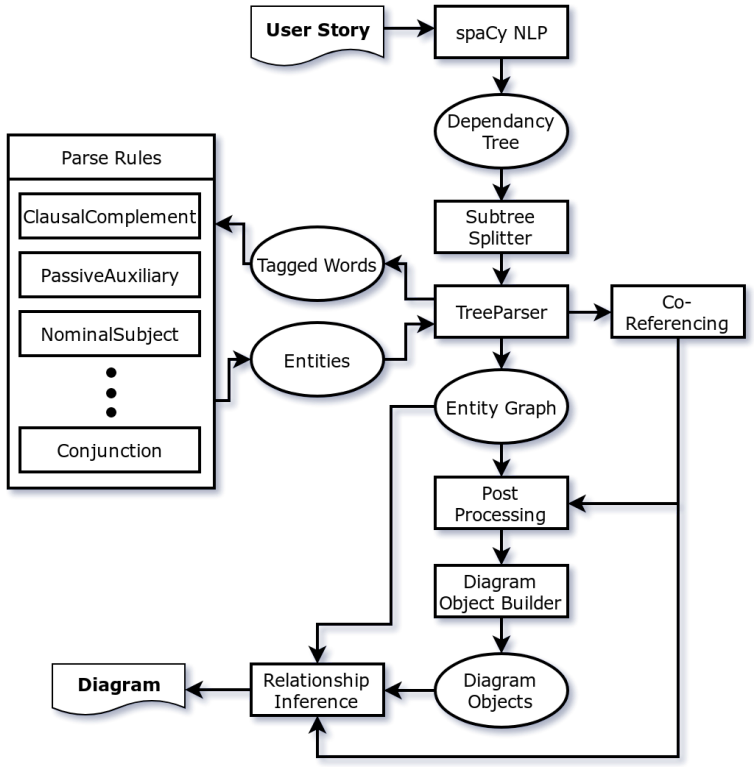


Figure 5.2: Diagramme de workflow de la Pipeline du projet Legacy [Gilson and Irwin, 2018])

### 5.2.3 Limitations

Le nombre limité et la simplicité des US utilisées lors de cette première phase de développement a permis de démontrer la faisabilité sur des cas simples mais un certain nombre de problèmes subsistent. Nous allons décrire ci-dessous les plus importants.

#### Découpe de l'histoire d'utilisateur

Pour identifier les différentes parties d'une US, chaque branche de l'arbre des dépendances générée par SpaCy est analysée pour identifier si celle-ci correspond à une des parties importantes. Pour ce faire, ils vont tester si certains mots clefs sont présents :

- *part-<acteur>* : *"as"* et *"a"* ou *"an"*
- *part-<objectif>* : *"I"* et *"Want"*
- *part-<bénéfice>* : *"so"* et *"that"*

Cette approche naïve entraîne divers problèmes. Le premier découle de l'hypothèse erronée selon laquelle SpaCy divise l'US en trois branches correspondant à ses trois différentes parties. Dans plusieurs situations, l'arbre est composé de plus de trois branches qui ne correspondent pas exactement à la découpe escomptée, ce qui entraîne une perte d'informations des branches non utilisées et différents problèmes de parsing. Le second est le manque de flexibilité. L'ensemble des cas n'est pas pris en compte comme l'utilisation d'un autre verbe que *"want"*, par exemple *"would"*, ...

#### Conjonction

L'utilisation des entités composites pour représenter une conjonction fonctionne dans des cas simples où il n'y a que deux éléments mais si celle-ci est composée de plus, l'approche ne fonctionne pas et il n'y a pas de prise en compte des virgules ou d'autres éléments pour représenter des conjonctions. Exemple fonctionnel :

*"As user, I want to create and delete a profile"*

Exemple Limite :

*"As user, I want to create, modify and delete a profile"*

#### Flux de contrôleurs

Dans beaucoup d'US, nous avons une suite d'actions dépendant les unes des autres.

*e.g., "As user, I want to create a page to delete my profile"*

où l'on peut constater que *"delete"* dépend de *"create"*. Les différentes dépendances entre action n'ont pas été prises en compte.

#### Relation de composition pour *"of"*

Quand deux *Entity* sont reliées par une préposition, cette relation doit être représentée par un lien de dépendance. Le *"of"* est une exception à cette règle. Il est la seule préposition qui crée une relation de composition entre les deux objets.

*e.g., "a dataset of my website"*

*"dataset"* est un élément qui compose *"website"*. Ce cas particulier fréquent dans les US n'est pas correctement traité.

#### Objet du diagramme seul ou mauvais candidat

Les US utilisées pour définir les règles du parseur de Legacy étaient peu nombreuses et très simples. Cela a causé la création de règles erronées ou l'oubli de certaines correspondant à des structures de phrases non observées ou mal interprétées. Ces problèmes entraînent pour les US moins triviales, la modélisation d'objets non-reliés ou erronés et l'oubli de certains d'entre eux. Nous pouvons donc déplorer une perte d'informations ou une mauvaise modélisation.

### 5.2.4 Modifications techniques

Dans ce point, nous allons aborder toutes les modifications techniques que nous avons effectuées sur les objets clés et certaines parties de la pipeline présentée précédemment. Les étapes ajoutées à la pipeline et la redéfinition des différentes règles de parsing ne seront pas abordées dans ce point mais le seront dans le prochain chapitre. Certaines de ces modifications viennent directement répondre aux limitations du point précédent.

#### Découpe de l'histoire d'utilisateur

Comme nous l'avons expliqué précédemment, une US peut être divisée en trois parties principales (*part-<acteur>*, *part-<objectif>*, *part-<bénéfice>*). Il est important de les identifier avant le parsing car nous devons traiter chacune d'elles différemment. La première sera la seule à pouvoir créer des *Actor*, la deuxième permettra d'identifier les autres éléments et la troisième sera ignorée car elle ne doit pas être modélisée dans le diagramme de robustesse. L'implémentation actuelle de cette découpe se base sur les branches identifiées par SpaCy. Comme expliqué précédemment, cela cause certaines limitations (des pertes d'information, mauvaise identification,... ).

Nous avons donc repensé cette étape en ne nous basant plus sur l'arbre des dépendances mais sur la phrase en elle-même. Nous avons employé une méthode beaucoup plus naïve où nous identifions les parties en fonction des premiers mots qui les composent. Premièrement, nous vérifions s'il y a une troisième partie en cherchant un *"so that"*. Si elle est présente, nous l'extrayons de l'US. Ensuite, nous identifions la deuxième partie à l'aide du premier *"I"* trouvé dans la phrase. Le reste correspond à la partie *"As"*. Nous avons testé notre approche sur l'ensemble de notre base de données et contrairement à l'approche précédemment utilisée, elle fonctionne pour l'ensemble des US respectant le template de Cohn [Cohn, 2004] en ne perdant pas d'informations.

#### Relation de composition du *"of"*

Avec l'implémentation de Legacy, une propriété n'est pas considérée comme un type différent d'entité mais comme un autre type d'objet. Cela pose un problème lorsque nous allons définir une *Property* composée d'un objet et d'extension qui se rapporte à celui-ci. Ce problème a lieu lorsqu'un *"of"* relie deux *Entity*.

*e.g., "big database of my small website"*

Nous voulons que *"big database"* soit la *Property* qui compose *"small website"*. Pour permettre à notre Objet *"database"* d'être traité comme un Objet sauf lors de l'étape de modélisation finale où il

devra être représenté comme une *Property*, nous avons ajouté un attribut aux entités parsées et aux objets du diagramme. Cet attribut sera un booléen qui sera vrai si notre *Entity* doit être considérée comme une *Property* et faux si non. Si cette *Entity* est fusionnée avec une autre n'ayant pas cet attribut vrai, elle ne devra plus être modélisée comme *Property*.

#### Nettoyage de la description des objets du diagramme

Dans certaines situations, nous avions deux DO avec des descriptions très similaires à l'exception de quelques mots partagés mais avec des accords différents (conjugaison, pluriel, ...). Pour éviter ces doublons, nous avons transformé l'ensemble des mots composant la description des DO en leur lemme. Nous avons aussi retiré l'ensemble des Stop-words de leurs descriptions car ces informations sont du bruit inutile. Ces deux étapes permettront d'unifier plus facilement nos DO et simplifieront leur lecture. Nous n'appliquerons pas ce processus sur les *Control* car nous préférons garder l'information intégrale et éviter des fusions abusives.

Actuellement, pour créer la description d'un *Control*, nous nous basons sur celles de ses enfants. Cela entraîne une répétition abusive de certains concepts et parfois celle-ci est très longue. Pour résoudre ces problèmes, nous avons tenté de retirer l'ensemble des descriptions des enfants modélisés. Cela a effectivement réduit la longueur du texte et a supprimé les répétitions. Malheureusement, ces suppressions l'ont rendu trop vague; dans la majorité des cas, nous n'avions que le verbe principal, ce qui posera un problème lors de la fusion des *Control* ayant une description commune. Sur base de cette observation, nous avons pu conclure que la description un peu trop fournie d'un *Control* est importante pour permettre sa bonne compréhension mais qu'il reste du travail pour trouver le juste milieu pour recréer ces descriptions (parfois très verbeuses ou grammaticalement incorrectes).

#### Création de l'arbre des dépendances

Le parseur implémenté dans la version précédente n'utilise pas la force de l'orienté objet proposé par les Tokens de SpaCy. Son auteur a recréé un arbre des dépendances composé d'objets Noeud définis par ses soins. Pour le générer, il va d'abord exporter l'arbre généré par SpaCy sous forme de JSON. Ensuite, il va naviguer à travers ce JSON pour créer les différents Noeuds.

Cette démarche n'était pas nécessaire, il aurait suffi d'appliquer directement notre parseur sur les Tokens SpaCy. Certaines informations précieuses comme la position d'un mot, etc ne sont pas contenues dans le JSON. Nous avons donc modifié la création de l'arbre en ne la basant plus sur le parsing du JSON, mais en naviguant directement à travers les Tokens pour le créer. Cela rend l'approche plus simple et permet d'accéder facilement à tout le potentiel de SpaCy. Il aurait été trop coûteux en temps de remplacer tous les appels à l'arbre des dépendances par des appels aux objets SpaCy.

## 5.3 Conclusion

Le projet Legacy nous propose une première version d'un prototype permettant de modéliser des histoires d'utilisateur sous la forme de diagrammes de robustesse. Sur des cas simples, celui-ci est efficace, mais pour les US plus complexes, les diagrammes générés sont souvent faux. Le manque de documentation du projet a rendu difficile la compréhension des différentes étapes qui le composent et des potentielles causes de ses problèmes. Toutefois, nous avons pu identifier un ensemble de cas posant problème *e.g.*, flux de *Control*, relation de composition du *"of"*, conjonction, ... Dans le prochain Chapitre, nous expliquerons comment nous avons résolu un grand nombre d'entre eux à l'aide de notre module de préprocessing et de la redéfinition d'un grand nombre des règles de parsing.

CHAPTER 6

# GÉNÉRATION DE DIAGRAMMES DE ROBUSTESSE DEPUIS DES HISTOIRES D'UTILISATEUR

*Dans ce troisième chapitre de contribution, nous allons expliquer les différentes modifications et ajouts réalisés au projet Legacy ("From User Stories to Use Case Scenarios Towards a Generative Approach") [Gilson and Irwin, 2018] exposé en détail dans le chapitre précédent. La première section présentera nos données et notre méthode de travail pour améliorer Legacy. Dans la seconde section, nous expliquerons en détail la nouvelle Pipeline, l'ensemble des étapes du Préprocessing, les règles de parsing qui composent notre parseur et les modifications effectuées sur la génération de diagramme de robustesse. Dans la troisième section, nous présenterons les différentes visualisations que nous proposons à l'aide du diagramme de robustesse. Dans la quatrième section, nous présenterons notre méthode d'évaluation et nos résultats. Dans la dernière section, nous discuterons de notre technique et de ses limitations.*

## 6.1 Méthode de travail

### 6.1.1 Description

La base de données que nous avons utilisée est la même que pour l'identification de QA présentée au Chapitre 4. Pour rappel, celle-ci est composée de 1 675 différentes US réparties entre 22 backlogs [Dalpiaz, 2018]. Chacun d'eux a été rédigé dans le cadre d'un projet industriel ou universitaire par des développeurs expérimentés et par des équipes différentes.

Contrairement à l'étape de développement précédente (Legacy), la base de données que nous utilisons nous permettra d'explorer beaucoup plus de types de phrases et de techniques de rédaction grâce à sa diversité d'écrivains et de domaines. Nous pourrons aussi découvrir des cas moins triviaux encore inexplorés.

### 6.1.2 Exploration

Pour mieux comprendre les limitations du projet Legacy, nous avons testé la génération du diagramme de robustesse d'un ensemble d'US. Sans surprise, les plus simples étaient pour la plupart correctement gérées mais un grand nombre d'entre elles moins naïves n'étaient pas bien modélisées. Nous avons donc pu confirmer les limitations citées dans le chapitre précédent.

Cette exploration nous a permis de découvrir d'autres problèmes découlant d'erreurs d'identification et de génération de l'arbre des dépendances par SpaCy. Comme nous l'avons expliqué dans l'état de l'art (Section 1.1.2), la génération de l'arbre des dépendances n'est pas fiable à 100%. Ces erreurs s'expliquent souvent par des tournures de phrase peu claires ou complexes. Pour notre cas, nous avons pu identifier d'autres causes liées à la façon dont sont écrites certaines US. Pour faciliter l'identification des mots clefs dans une US, certains développeurs ont pris l'habitude d'écrire leur premier caractère en majuscule, ce qui pousse SpaCy à les identifier comme des noms propres ou le début d'une nouvelle phrase. Une autre erreur d'écriture déstabilisant SpaCy est le placement d'une "," avant une conjonction de coordination *e.g., "I want to create , and use a site"*. Cela empêche la bonne identification de la relation de conjonction entre deux mots. Un dernier détail causant des problèmes pour la génération de l'arbre est l'utilisation d'espace multiple à la place d'un espace simple.

Une autre observation intéressante est que certaines US ne possèdent pas d'action mais juste le souhait d'avoir un objet *e.g., "..., I want a profile"*. Cette action manquante empêche la modélisation. Nous avons présenté précédemment les règles proposées par différents framework pour écrire de bonnes US (INVEST et Quality User Story Framework). Comme nous pouvions l'envisager, certaines US ne les respectent pas (plusieurs actions définies, plusieurs phrases en une, phrase trop longue et peu claire,...). Nous devons donc trouver des solutions pour permettre à un maximum de ces US mal écrites d'être tout de même le mieux modélisées possible.

### 6.1.3 Définition des règles de parsing

La méthode demandée par le Docteur F. Gilson pour redéfinir les différentes règles du parseur est une approche inductive. Premièrement, nous avons récupéré pour chaque règle l'ensemble des US ayant un mot répondant à la condition de celle-ci. Deuxièmement, nous avons analysé l'ensemble de ces exemples sur base de la phrase entière et les mots connectés au mot à parser par la règle. Troisièmement, sur base de ces observations, nous avons défini différentes classes où le parsing serait plus ou moins différent. En règle générale, pour définir ces classes, nous nous basons sur :

- le POS/TAG du mot en cours de parsing
- le type d'entité parent et/ou des entités ancêtres
- le POS/TAG et lien de dépendance des noeuds enfants

Pour quelques cas exceptionnels, nous devrons utiliser le lemme du mot, celui d'un ancêtre ou d'un enfant. Nous avons analysé l'ensemble de ces informations pour définir les meilleures classes. Il ne faut pas oublier que SpaCy commet des erreurs de détection qui auront dans certains cas des répercussions fâcheuses *e.g.,* identification d'un objet comme un verbe ce qui causera la création d'un *Control* à la place d'une *Entity*.

Pour évaluer la modification d'une règle, nous avons comparé les diagrammes de robustesse générés avant la modification et après pour l'ensemble des US ayant un mot répondant à la règle modifiée. Si le nombre de modélisations considérées comme plus proches de la modélisation correcte est plus important que le nombre de celles moins proches, alors nous la validons. La même approche a été utilisée pour valider les phases de pré-processing.

## 6.2 Nouvelle Pipeline

### 6.2.1 Vue d'ensemble de la Pipeline

La Figure 6.1 nous présente la nouvelle pipeline du projet. Les grands changements par rapport à celle du projet Legacy sont :

- l'ajout d'un module de préprocessing au début de la pipeline
- la redéfinition d'un grand nombre de règles de parsing
- la fusion des DO similaires
- la génération de différents diagrammes de robustesse sur base d'un même backlog en fonction des besoins de l'utilisateur

### 6.2.2 Pré-processing

La phase d'exploration nous a permis de découvrir un ensemble d'erreurs récurrentes causant des problèmes lors de notre parsing. Pour en résoudre une partie, nous avons opté pour l'ajout d'un module de pré-processing dans la Pipeline. Celui-ci se place à la première étape. Son objectif est de nettoyer l'US, d'ajouter les actions manquantes et de la simplifier en la divisant en sous US plus simples pour mieux respecter les règles vérifiant la qualité syntaxique proposées par le Quality User Story Framework. Chacun des choix a été testé sur l'ensemble de la base de données et la décision a été prise en fonction des observations.

#### Nettoyage

Cette première étape a pour but de retirer les problèmes d'écriture causant des difficultés à SpaCy pour effectuer une bonne génération de l'arbre des dépendances et une bonne identification des TAG/POS. Pour chacune des difficultés, nous allons expliquer notre solution :

- Remplacer les espaces multiples par des espaces simples.
- Remplacer la première lettre en majuscule par sa minuscule pour tous les mots composés de plus d'un caractère et ayant une seule majuscule. Nous avons choisi cette option après avoir testé le fait de mettre l'entièreté de l'US en minuscule mais cela a causé d'autres soucis à SpaCy. Les seuls risques de notre option est la mauvaise détection de noms propres mais pour la majorité des cas, ils resteront classifiés comme des noms, ce qui ne nous pose pas de problème.
- Supprimer les acteurs inconnus dans la deuxième partie de la phrase.

  *e.g., "As user, I and my team want to create . . . "*

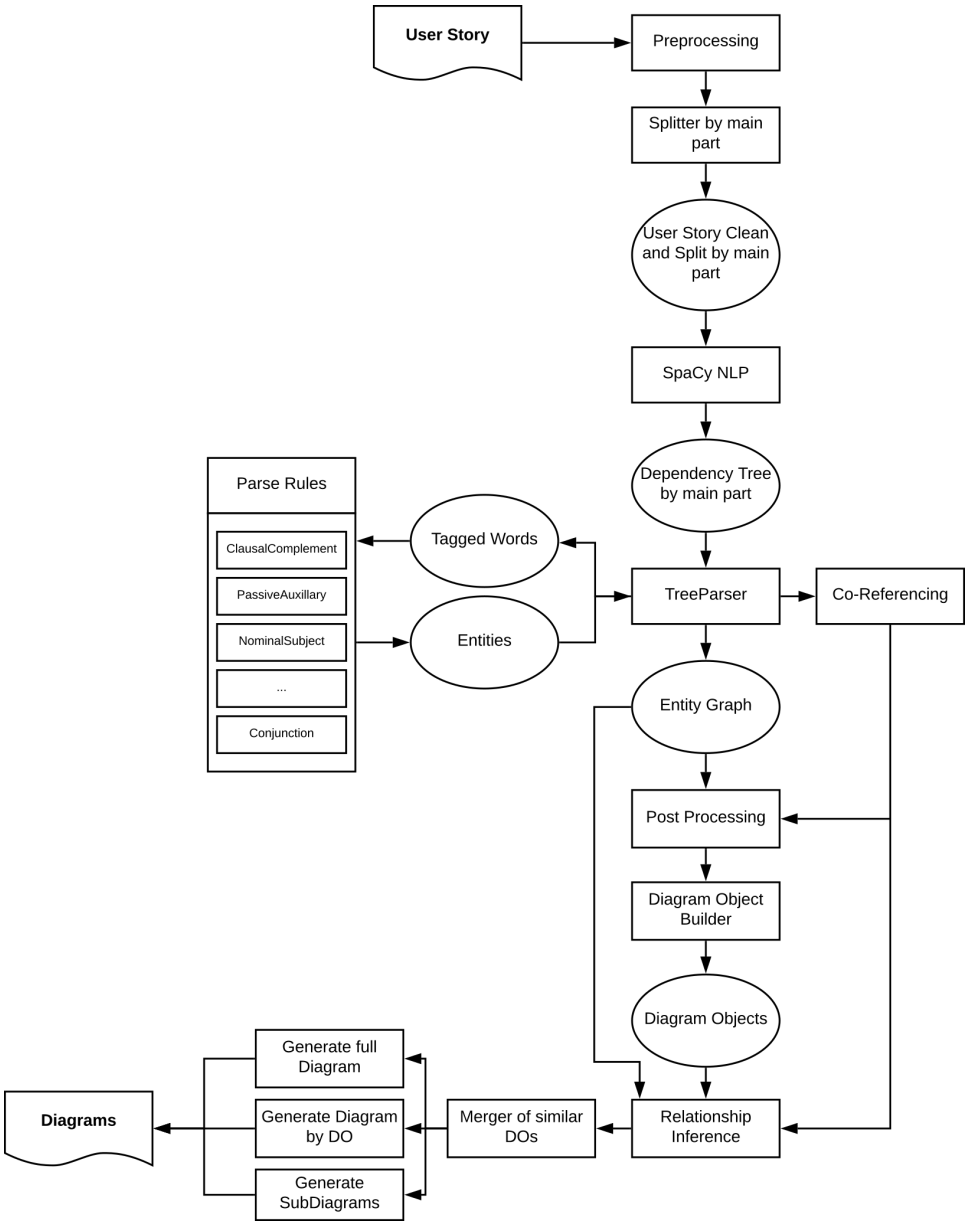


Figure 6.1: Diagramme de workflow de notre nouvelle Pipeline pour la génération de diagrammes de robustesse à partir d'histoires d'utilisateur

Nous devons retirer *"my team"* qui n'a pas été défini dans la première partie. C'est une erreur d'écriture.

- Remplacer les "/" par des *"or"*. Nous avons opté pour cette décision après avoir essayé de

définir une règle plus complexe sur base du type des mots de part et d'autres du "/". Malheureusement, aucune règle n'est définissable et SpaCy n'identifie pas toujours correctement le mot qui précède comme étant une conjonction. Le nombre de cas où c'est une conjonction est beaucoup plus élevé ; donc, nous optons pour un remplacement par défaut des "/" par "or". Les conséquences de cette règle est la suppression des "/" dans le diagramme et la mauvaise identification lorsque "/" n'est pas employé comme une conjonction de coordination.

- suppression des "," avant une conjonction de coordination

*e.g., "...I want to use ,and to add . . . " => "...I want to use and to add . . . "*

**Action manquante**

Comme nous l'avons mentionné précédemment, certaines US ne possèdent pas d'action, ce qui cause un problème majeur pour une bonne modélisation. En analysant toutes les US ayant une action manquante, nous avons pu constater que pour chacune d'elles, l'acteur souhaite avoir un objet. Nous avons donc opté pour l'ajout d'une action par défaut qui sera *"to be provided with"*

*e.g., "... I want a profile ..." => "... I want to be provided with a profile ..."*

**Décomposition des conjonctions d'action**

L'un des problèmes les plus complexes dans le parsing est la gestion des conjonctions. Pour le simplifier, nous avons opté pour la solution suivante : diviser une US ayant des conjonctions dans ses actions en sous-US. Nous avons pu identifier deux cas différents : les actions partagent le même complément ou pas. Pour documenter nos règles, nous utiliserons P pour partie de phrase, VB pour verbe et CC pour une conjonction de coordination. Le premier cas où les actions partagent un complément commun : nous allons donc identifier les actions et pour chacune d'elles, nous l'entourerons du reste de l'US à l'exception de la conjonction de coordination et de l'autre action.

*"P1 VB1 CC VB2 P2" = > ["P1 VB1 P2",P1 VB2 P2 ]*

*e.g., "As User, I want to create and use a profile" => "As User, I want to create a profile" et "As User, I want to use a profile"*

Nous devons aussi prendre en compte le cas où nous avons deux sous-ensembles d'actions.

*e.g., "... I want to create and add a page to use or move a profile ..."*

Nous allons gérer chacun des sous-ensembles indépendamment.

Le deuxième cas où les actions ne partagent pas le même complément : nous devons identifier les actions et leurs compléments puis nous les entourerons du reste de l'US à l'exception de la conjonction, des autres actions et leurs compléments.

*"P1 to VB1 cc VB2 P2" => ["P1 to VB1 P2","P1 to VB2 P2"]*

Comme pour l'autre cas, nous traitons les sous-ensembles d'actions de la même manière.

Le problème observé avec cette technique est la disparition d'un lien de co-référencement entre les deux compléments.

*e.g.,. "... I want to create some profiles and use them ... " => "... I want to create some profiles... " et "... I want to create use them ... "*

Le *"them"* perd sa référence à *"profile"*. Nous allons donc essayer de le permuter au préalable par le mot qu'il remplace à l'aide d'une solution d'identification des co-référencements.

#### Co-référencement

Naturellement, certains mots sont remplacés par des pronoms pour éviter les répétitions abusives dans nos US.

*e.g., "As an archivist, I want to search **images** by people represented in **them**"*

Nous avons donc cherché une solution à ce problème car pour un humain, il est simple de créer le lien entre un pronom et le mot à qui il fait référence mais pour notre parseur, cela est beaucoup plus compliqué. La solution que nous envisageons est le remplacement des pronoms par le mot auquel ils font référence.

Malheureusement, comme expliqué dans l'état de l'art (Section 1.1.2), le problème de co-référencement reste l'un des plus gros challenges du NLP et les solutions actuellement proposées ne nous donnent que des résultats assez médiocres (avec un *F1-Score* de 74%) en comparaison de de ceux pour l'identification du POS/TAG (avec un *F1-Score* supérieur à 97%). Nous avons tout de même pris la décision d'explorer les résultats offerts par la Librairie Python Neuralcoref basée sur SpaCy pour notre cas. Nous avons exploré les différents mots qu'elle considère comme une Co-ref et nous avons pu constater qu'elle ne se contente pas de détecter les pronoms mais aussi les déterminants *e.g., "my", "his", ...* que nous ne voulons pas remplacer. Nous avons donc retiré tous ces cas de notre analyse. En explorant les résultats pour les pronoms, nous avons pu constater qu'un grand nombre n'était pas détecté, mais pour ceux qui le sont, le mot de référence indiqué est très souvent le bon. Nous nous limiterons aux pronoms de la $3^{ème}$ pers (*"him","them","it"* et *"her"*) qui sont les seuls éléments qui nous intéressent pour la modélisation car ceux-ci peuvent remplacer des *Entity*. Les pronoms *"it"* et *"her"* ne sont pas uniquement utilisés pour remplacer un objet, *"it"* peut être un sujet et *"her"* un pronom possessif. Nous voulons éviter le remplacement dans ces cas ; donc nous allons ajouter une contrainte vérifiant que Spacy les identifie comme des objets.

En évaluant cette approche, nous avons pu constater qu'elle ne détectait pas l'ensemble des problèmes de co-référencement mais ceux détectés apportent une meilleure modélisation finale. Cette étape pourra certainement être améliorée avec les avancées futures de l'état de l'art.

### 6.2.3 Parseur

Pour notre parseur, nous nous sommes basés sur celui déjà existant dans Legacy décrit dans le Chapitre 5. Notre tâche était d'étendre/modifier les règles actuelles pour qu'il soit capable de gérer des cas plus complexes. Etant donné le manque de documentation, la tâche de compréhension de chacune des règles a été très complexe et même impossible pour certaines. Pour éviter une

modification causant des dommages importants sur les résultats, nous avons travaillé de façon expérimentale. Pour chacune de nos modifications, nous sélectionnions l'ensemble des US affectées par ce changement et nous avons comparé les diagrammes générés avant et après le changement.

Ci-dessous, nous présentons l'ensemble règles de notre parseur. Pour chacune, nous définirons les différentes classes qui la composent, s'il y en a plus d'une et nous expliquerons notre décision. Pour les plus complexes, nous définirons brièvement le type de dépendance. Nous expliquerons la démarche employée pour les définir sur base de nos observations.

Pour mieux comprendre l'action des règles de parsing, nous avons défini ci-dessous une manière de modéliser l'application d'une règle. Dans notre notation, nous utiliserons les formes pour différencier les entités parsées (oval) par rapport aux noeuds de l'arbre des dépendances (rectangle). Le noeud en cours de parsing sera représenté dans la couleur bleue et la ligne du rectangle l'entourant sera discontinue. L'entité Parent pour la prochaine règle sera représentée en vert et ses traits seront en gras. Nous représenterons les relations d'enfant vers parent à l'aide d'une flèche continue. Nous modéliserons la relation entre l'entité Parent donnée en entrée de la règle et le token en cours de parsing par une flèche discontinue. Nous pouvons retrouver à la figure 6.2 les notations utilisées pour nos modélisations. Pour distinguer clairement les différents éléments, chacun sera muni d'un identifiant :

- **P** = entité Parent pour la règle
- **GP** = Grand Parent, entité parent du Parent
- **A** = Ancêtre, entité ancêtre la plus proche du Parent qui respecte une condition
- **T** = Token en cours de parsing
- **NE** = Noeud Enfant du token en cours de parsing

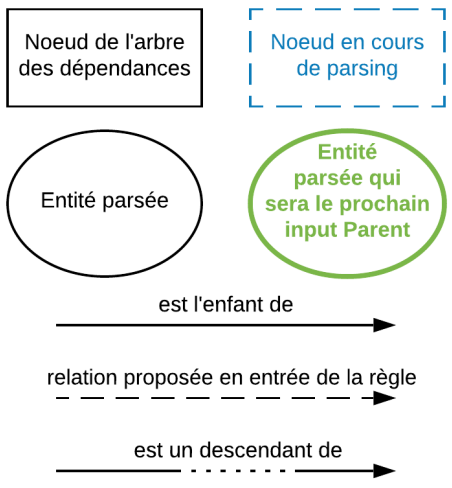


Figure 6.2: Notation utilisée pour modéliser l'application d'une règle de parsing

Il est important de préciser que nous ne modélisons pas l'ensemble des éléments à un moment du parsing d'une US, mais seulement ceux nécessaires par la règle. La modélisation est découpée en deux parties. La partie gauche correspond à l'état avant l'application de la règle. Sur la partie supérieure, nous retrouvons les entités parsées ancêtres et sur la partie inférieure, nous avons les noeuds enfants dans l'arbre des dépendances. La partie droite présente la nouvelle entité parsée et sa position par rapport aux autres entités.

Pour décrire les conditions nécessaires à l'application de notre règle, nous utiliserons la notation:

- **.TAG** : pour le TAG d'un token
- **.POS** : pour le POS d'un token
- **.dep** : pour le type de dépendance entre ce token et son token père
- **.lem** : pour le lemme d'un token ou d'une entité

Par exemple à la Figure 6.3, nous modélisons une règle qui exige de notre Token que son TAG soit égal à VBG ou VBN, qu'au moins un de ses enfants dans l'arbre des dépendances ait son lemme égal à « to » et que l'entité Parent proposée en entrée soit une Relation. Si cette condition est validée, le Token sera parsé et il deviendra une entité de type Relation qui sera un enfant de l'entité Parent.

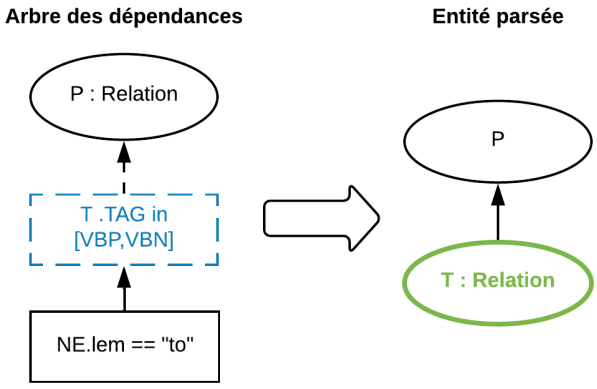


Figure 6.3: Exemple de visualisation d'une règle de parsing

### Simple relation

Nous pouvons tout d'abord trouver un ensemble de règles simples qui transformeront notre token en une Entité et qui la lieront à leur père. Les seules variantes sont au niveau du type de l'Entité créée.

Ce premier ensemble de règles regroupe les mots qui n'ont pas pour vocation de devenir des objets indépendants de notre diagramme mais qui viendront seulement compléter la description de leur Parent, si nécessaire. Pour cela, ils vont être convertis en Entité de type extension qui sera

connectée au Parent. Les types de mots affectés par cette règle sont les AdjectivalComplement, les Attribute, les PhrasalVerbParticle, Preposition et les Dative (Visualisation à la Figure 6.4).

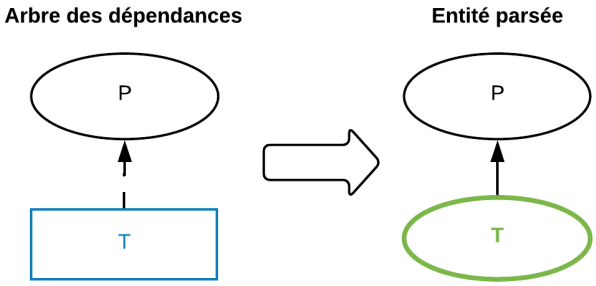


Figure 6.4: Simple règle de parsing transformant le Token en entité Extension

Le deuxième ensemble regroupe des actions. Celles-ci vont être transformées en Entité de type Relation et elles vont être connectées au Parent. Nous pouvons retrouver les mots étant des ClausalModifier ou des OpenClausalComplement (Visualisation à la Figure 6.5).

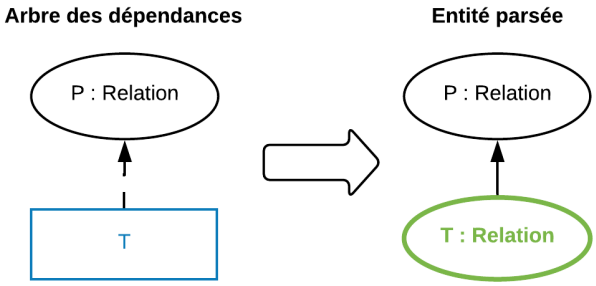


Figure 6.5: Simple règle de parsing transformant le Token en entité Relation

#### Information supplémentaire

Il existe un ensemble de mots qui sont purement et simplement des informations supplémentaires pour une entité et qui ne pourront pas devenir une propriété par la suite. Celles-ci sont les Case, GenericContext, les PassiveAuxiliary et GenericModifier. Chacune d'elles correspond à un type d'information utile pour la création d'une description plus précise d'un élément. Pour ce cas, nous ne créons pas de nouvelle entité; le mot sera simplement ajouté dans l'un des attributs du

père. Pour le Case, ce sera dans case; pour les 3 autres cas, ce sera dans le prétexte du Parent (Visualisation à la Figure 6.6).

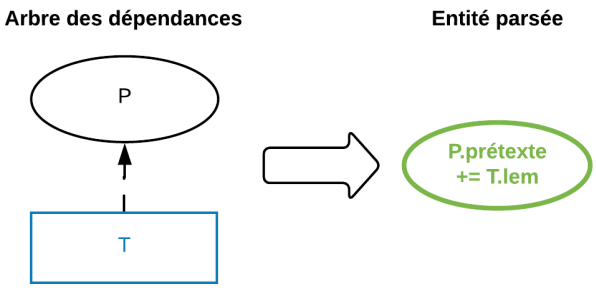


Figure 6.6: Simple règle de parsing ajoutant le lemme du Token au prétexte du Parent

#### Négation

Les mots représentant la négation comme *"not", "no"*, etc sont des informations importantes pour la bonne compréhension du sens d'une action. Ils vont donc être transformés en entité de type Extension et liés au Parent pour pouvoir apparaître dans la description de ce dernier. La seule exception relève des négations compressées avec un autre mots *e.g., " I don't "*, que nous ajouterons simplement au prétexte du Parent qui correspond au mot auquel elle est fusionnée.

#### Composant

Parfois, pour nommer un concept, il est nécessaire d'utiliser plusieurs mots.

*e.g., "my email account"*

*"email"* est le composant de *"account"* permettant de faire référence à un nouvel objet. Les composants vont être ajoutés comme des propriétés à leur ancêtre le plus proche qui n'est pas une Extension car une propriété peut uniquement faire référence à de potentiels objets du diagramme. Il y a deux exceptions :

- si Parent est un Acteur, le token est seulement ajouté à la description de celui-ci ;
- si l'ancêtre le plus proche qui n'est pas une Extension possède le même lemme que notre token, nous ajoutons notre composant à la description du Parent et nous le convertissons en une Interface (Figure 6.7).

#### Sujet

Cette règle s'applique sur les sujets d'un verbe. Elle n'a pas changé depuis le Legacy. Les développeurs ont d'abord catégorisé les sujets en sujets directs et en sujets indirects (mot remplacé par un pronom). Sur base de cette découpe , ils ont défini différentes façons de parser le token :

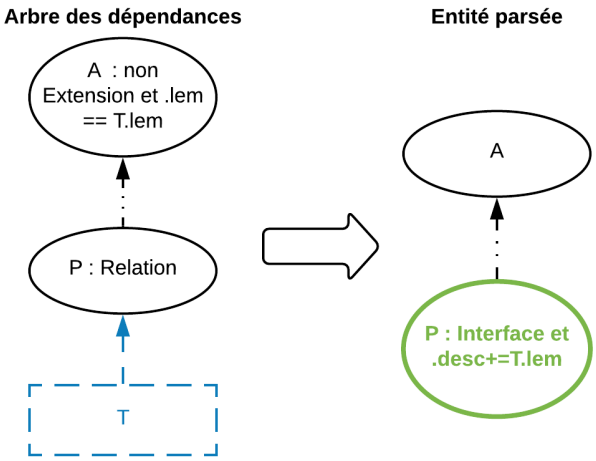


Figure 6.7: Règle de parsing pour la deuxième exception de Composant

- si le token est un sujet nominal et indirect, nous créons une Entité de type Primary-actor-proxys qui permettra la création de liens entre les Acteurs et les pronoms qui y font référence.
- si le token est un autre type de sujet et indirect et si un sujet de ce type a déjà été vu précédemment dans l'US, alors nous dupliquons l'Entité de celui-ci. Sinon, nous créons une Entité de type Object-proxys qui permettra la formation de liens entre les Entités et les pronoms qui leur font référence.
- si le token est un autre type de sujet et direct, nous créons une Entité du type par défaut donnée en paramètre (Acteur ou Extension) sauf si le *Carryover* nous indique que c'est une Interface. Nous l'ajouterons aux références du *Carryover*.

Pour chaque situation, nous lions l'entité créée au Parent.

**PossessiveNominalModifier**

Les modifieurs nominaux possessifs sont les mots permettant de marquer l'appartenance d'une chose à une autre. Nous pouvons observer deux façons de faire :

- l'utilisation d'un pronom possessif *e.g., "my page"* ; le *"my"* permet de dire que *"page"* m'appartient. Dans ce cas, nous transformons le token en une entité de type primary-actor-proxy.
- avoir recours au *" 's"* à la fin d'un mot *e.g., "organization's documents"* : grâce au *"'s"* nous pouvons comprendre que les documents appartiennent à l'organisation. Dans cette situation, nous créons une entité de type Extension (Comme à la Figure 6.4).

Dans les deux cas, l'entité créée deviendra un enfant du Parent.

**AdverbialClausalModifier**

Un AdverbialClausalModifier est un groupe de mots qui dépend d'un adverbe. Ce genre de dépendance était ignorée dans la version précédente, mais nous avons pu observer que dans certaines circonstances, elle pouvait apporter des informations devant être modélisées. En explorant les différents TAG, nous constatons que le token identifié comme AdverbialClausalModifier est un verbe (les autres TAG correspondaient à des erreurs de la part de SpaCy).

Pour mieux comprendre la nature et l'utilisation de ces mots, nous avons récupéré l'ensemble des occurrences où cette règle est présente et nous avons identifié les différents TAG pour chacun. Nous avons observé 9 types différents (NN, NNP ,VBP ,VBG ,VB ,VBZ ,VBN, IN, JJ). Nous avons exploré chacune de ces catégories :

- **VB** : ce sont des verbes à l'infinitif qui suivent une autre action.

  *e.g., "As a site visitor, I want to click the link from the article teaser to take me directly to the body of the article."*

- **VBP** : ce sont des verbes au présent qui ne sont pas à la $3^{ème}$ personne du singulier et ils sont souvent précédés d'un marqueur, *e.g., "if" , "when"*, ou d'un *"to"*.
- **VBG** : ce sont des verbes au gérondif (forme "ing") qui viennent ajouter de l'information à une action ou à un objet.

  *e.g., "As a moderator, I want to log in using my account name and password"*

- **VBZ** : ce sont des verbes au présent à la $3^{ème}$ personne du singulier et comme pour VBP, ils sont souvent précédés d'un marqueur, *e.g., "if" , "when"*.

  *e.g., "As an OlderPerson, I want to receive a message when my home care nurse is on her way."*

- **VBN** : ce sont des verbes au participe passé. Comme pour VBP, ils sont souvent précédés d'un marqueur, *e.g., "if" , "when"* ou d'un *"to"*.

  *e.g., "As a site admin, I want to be emailed whenever a job is submitted, so that I am aware of it and can decide if I want to post it."*

- Pour les autres catégories, nous avons pu observer qu'elles étaient soit des erreurs de parsing, doit des parties de phrase inutiles pour notre modélisation.

Nous pouvons identifier une première classe composée des verbes à l'infinitif qui correspondront toujours à de simples actions. Nous les transformerons en entité Relation et nous les connecterons au Parent (Voir la Figure 6.8).

La seconde classe regroupe les verbes au gérondif (forme -ing) qui apportent des informations supplémentaires à leur parent. Nous allons donc créer une entité de type Extension et la lier au Parent (Même visualisation qu'à la Figure 6.4 en ajoutant comme contrainte que le Token doit être au gérondif).

Pour les classes restantes, le verbe devra être au VBP ou au VBN. Pour la troisème classe, notre token doit être précédé d'une Mark, *e.g., "if", "when"*, et son sujet doit être de la première personne. Ce cas correspond à une précondition nécessaire pour que l'action précédemment citée soit possible. Nous devons donc créer une entité Relation qui sera placée avant son action Parent. Le parent du Parent deviendra celui de notre nouvelle entité et le Parent deviendra l'enfant (Voir la Figure 6.9).

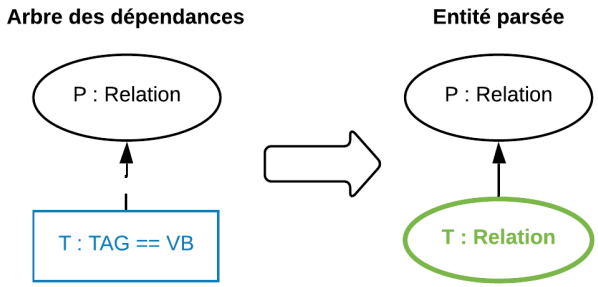


Figure 6.8: Règle de parsing pour la première classe d'AdverbialClausalModifier

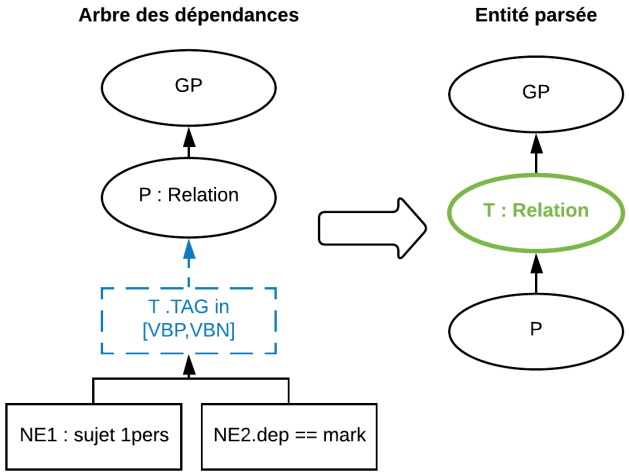


Figure 6.9: Règle de parsing pour la troisième classe d'AdverbialClausalModifier

Dans le quatrième classe, notre verbe doit être précédé d'un *"to"*. C'est une action qui suit celle précédemment citée. Nous aurons donc une entité Relation liée au Parent qui est du type Relation (Voir la Figure 6.10).

La dernière classe correspond au verbe n'ayant pas de sujet à la première personne, ni précédé d'un *"to"*. Celle-ci sera ignorée car elle représente des actions qui ne dépendent pas de l'acteur car notre sujet n'est pas à la première personne.

*e.g., "I want to create a profile if there **are** 3 others profiles"*

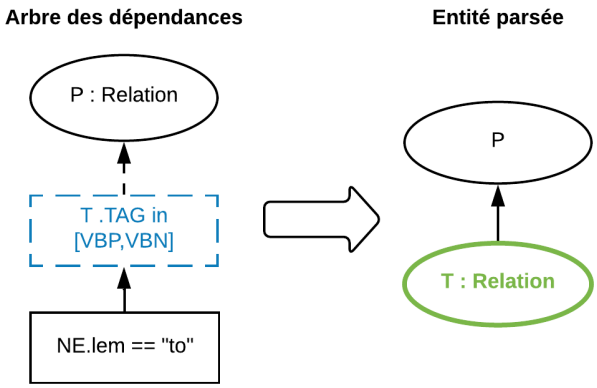


Figure 6.10: Règle de parsing pour la quatrième classe d'AdverbialClausalModifier

**ClausalComplement**

Une clause de complément est une clause introduite par un complementizer *e.g., "that"* ou *"whether"*. Une clause de complément est liée à un nom, un adjectif ou un verbe précédent. Au vu de sa nature, les éléments intéressants à analyser sont le complementizer utilisé et le type de mot auquel il se rapporte. Le TAG de notre mot n'apporte pas d'information intéressante et nous avons uniquement des verbes.

En explorant les complementizer, nous avons constaté une première classe où notre Clausal-Complement est introduit par un *"to"*. Nous avons pu identifier que cette première classe se rapporte toujours à un objet qui doit devenir l'interface de l'action du ClausalComplement. Nous allons donc créer une entité de type Relation liée au Parent pour notre Token et nous devrons transformer son sujet en entité de type Interface. Le sujet étant un enfant de notre token, nous allons informer le *Carryover* que le prochain sujet doit devenir une entité de type Interface.

*e.g., "As a user, I want the publish button in FABS to **desactivate** a page."*

*"Publish button"* est l'interface permettant l'action *"desactivate"* qui est la ClausalComplement. Nous devons donc créer une Entité Relation et la connecter au Parent.

Notre deuxième classe sera identifiée par le fait qu'elle est précédée d'une Mark *e.g., "if", "since"* et que son sujet est à la $1^{ère}$ personne. Etant donné que ce sont des actions à la première personne, cela signifie qu'elles sont accomplies par notre Acteur, donc nous les convertirons en entité Relation. Si nous avons un ancêtre de type Relation, la Mark exprime que l'action doit être accomplie avant celle de son ancêtre. L'ancêtre deviendra un enfant et le parent de l'ancêtre deviendra celui de l'entité actuellement parsée (Même visualisation que la troisième règle de parsing pour AdverbialClausalModifier à la Figure 6.9).

*e.g., "As a researcher, I want to see descriptive metadata for the item **whether I come to the item through the repository."***

L'ensemble des cas restant seront transformés en entité Extension car ils ne correspondent pas à un objet indépendant mais à de l'information complémentaire. Ils seront liés au Parent (Voir la Figure 6.4).

**Conjunction**

La technique précédemment utilisée consistait à créer des composants à partir des 2 éléments de la conjonction. Malheureusement, le manque de documentation la rend difficile à comprendre et à modifier. Ses résultats étaient assez limités ; beaucoup d'objets étaient oubliés. Les conjonctions formées de plus de 2 éléments n'étaient pas gérées. Ces multiples problèmes nous ont poussés à ajouter l'étape de pré-processing présentée précédemment pour la découpe des actions. Nous avons introduit une nouvelle façon de gérer les conjonctions. Pour modéliser une conjonction, nous voulons que les différents éléments qui la composent soient modélisés au même niveau.

*e.g., " add a profile and a page"*

Nous voulons que *"profile"* et *"page"* soient les enfants de *"add"* contrairement à l'arbre des dépendances qui place *"add"* comme enfant de "profile". Pour réaliser cela, il nous faudra explorer les ancêtres du Parent, (*"profile"* ) afin de trouver le plus proche qui est une entité Relation. Nous dupliquerons pour obtenir une Relation *"add a profile"* et une autre *"add a page"* (Visualisation de la règle à la Figure 6.11).

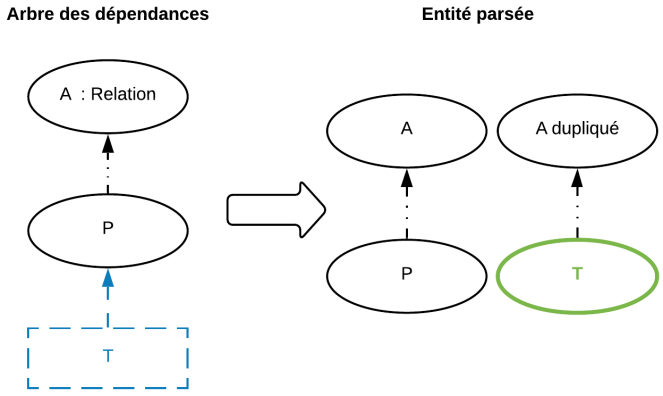


Figure 6.11: Nouvelle règle de parsing pour Conjonction

Cette approche a permis de gérer bien plus efficacement les conjonctions multiples. Les limitations de cette approche se trouvent à deux niveaux :

- elle nécessite un grand-parent. Si elle n'en n'a pas, nous utilisons l'ancienne méthode.;
- elle connectera les enfants du dernier élément de la conjonction seulement à ce dernier. *e.g.,*

*e.g., "add a profile and a page for my site"*

*"site"* sera uniquement connecté à page.

Pour la première limitation, étant donné que ces cas sont rares, nous avons opté pour l'utilisation de l'ancienne méthode si le grand-parent est manquant.

Pour la seconde, le problème est beaucoup plus compliqué à gérer. Une première hypothèse serait de partager l'ensemble des enfants du dernier élément aux autres éléments qui composent la conjonction. Nous observons une première limite dans le cas où le reste de la phrase n'a pas de lien avec les autres éléments de la conjonction. Si nous reprenons notre exemple, rien ne nous dit que le *"site"* doit absolument être connecté à *"profile"* ; c'est une déduction que nous avons faite. La seconde est lorsqu'un enfant se rapporte explicitement à notre élément final et pas aux autres

*e.g., "add a profile and an amazing page" => "add an amazing profile" et "add an amazing page".*

**PrepositionComplement**

Un complément par préposition est un mot lié à un autre par une préposition et qui a pour but de le compléter. Nous avons tout d'abord observé les différents TAG qui nous ont permis de voir qu'une PrepositionComplement peut prendre un grand nombre de formes. Une partie d'entre elles sont des verbes et donc de potentielles entités de type Relation.

Nous avons donc défini une première classe pour les verbes au gérondif (VBG) qui pour l'ensemble des exemples correspond à une action qui doit être accomplie avant celle à laquelle elle se rapporte. Notre token va donc être parsé comme une entité Relation et il aura comme parent le grand-parent actuel et deviendra le parent du Parent (Figure 6.12).

*e.g., "As an archivist, I want to search images **by uploading** an image."*

*"uploading"* est l'action qui doit se produire avant *"search"*.

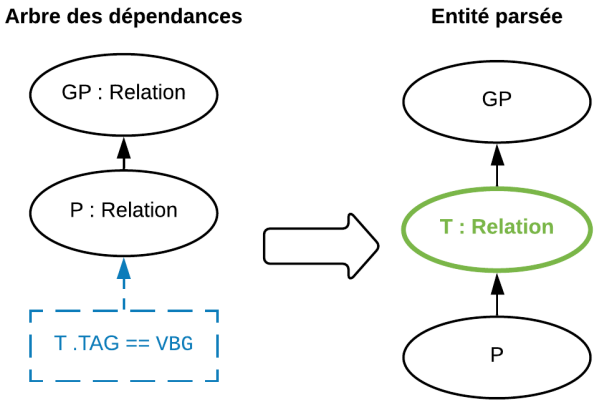


Figure 6.12: Règle de parsing pour la première classe de PrepositionComplement

Nous avons pu observer que pour les token qui ne sont pas repris dans la première classe, ceux-ci correspondent à un simple ajout d'information. Nous allons donc les transformer en entité Extension et les lier au Parent (Figure 6.4).

**RelativeClausalModifier**

Une clause relative est une expression descriptive se rapportant à un mot de la phrase principale. Celle-ci commence généralement par un pronom relatif *e.g., "who", "that"*. L'ensemble des occurrences de notre base de données étant des verbes, nous allons donc devoir décider entre Relation ou Extension.

La première classe que nous avons identifiée est celle où notre token est précédé d'un *"to"*. Nous pouvons la décomposer en deux cas : si notre token possède un ancêtre de type Relation ou non. Dans les deux situations, notre token sera transformé en entité Relation. Si nous avons un ancêtre de type Relation, celui-ci va devenir le parent de notre nouvelle entité (Figure 6.13).

*e.g., "As a user, I want to have a better place* ***to collect*** *project material.s"*

Cette situation est très intéressante parce qu'elle introduit une nouvelle action (*"collect"*) qui devra suivre son action parent (*"have a better place"*).

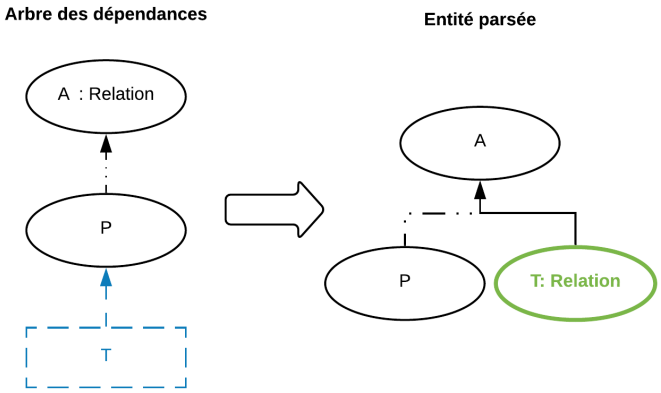


Figure 6.13: Règle de parsing pour la première classe de RelativeClausalModifier ayant un ancêtre de type Relation

Si nous n'avons pas d'ancêtre de type Relation, nous allons modifier le type du Parent en Interface car c'est par cet objet que l'acteur accomplira l'entité parsée par cette règle (Figure 6.14).

La seconde classe regroupe tous les token ayant comme token enfant un sujet à la $1^{ère}$personne. Nous avons pris la décision de les ignorer car ceux-ci correspondent à des préconditions trop vagues que pour être modélisées

*e.g., "As a Data Publishing User, I want to be able to delete a dataset* ***I have published****."*

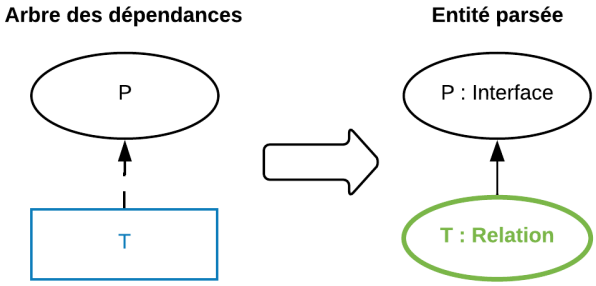


Figure 6.14: Règle de parsing pour la première classe de RelativeClausalModifier n'ayant pas d'ancêtre de type Relation

Ceux-ci représentent une action *"published"* qui a dû être faite par le passé pour accomplir l'action principale *"delete"*. On peut constater que ce genre d'action de type précondition est très vague et sa modélisation apporterait de la confusion inutile.

La dernière classe correspond aux autres cas. Elle est souvent introduite par un *"that"* ou *"which"*.

*e.g., "As a repository administrator, I would like to be able to continue to provide access to the repository in the event* ***that the server fails****."*

Elles sont composées d'informations pour l'entité à laquelle elles se rapportent. Nous allons donc créer une entité de type Extension et la lier au Parent (Figure 6.4).

#### GenericObject

Les objets génériques regroupent l'ensemble des mots que SpaCy identifie comme étant un objet. Ce sont très souvent des noms communs ou propres. Cette règle est restée presque inchangée car malgré son manque de clarté, elle obtient des résultats satisfaisants. les créateurs de Legacy vont d'abord modifier l'arbre des dépendances pour s'assurer que le token n'ait qu'un Componant comme enfant direct ; les autres seront placés comme enfant de ce Componant et ainsi de suite de telle sorte que chaque Componant n'ait qu'un enfant de type Componant. Notre token va être converti en une Entité ayant le type par défaut (Acteur ou Extension).

Si notre mot est utilisé pour atteindre une action, il deviendra une Interface. Comme pour les sujets, le *Carryover* va être utilisé pour retenir les objets employés et permettre le lien entre les pronoms y faisant référence.

Un cas particulier n'était pas géré, celui du *"of"* signifiant la relation de composition d'un objet vis-à-vis d'un autre. Dans Legacy, il était traité comme une relation de dépendance.

*e.g., "As admin, I want to use the database of my site"*

l'objet *"database"* est un élément faisant partie de l'objet *"site"*. Pour ce faire, nous avons introduit un attribut supplémentaire aux entités parsées permettant de dire qu'un Objet est une propriété d'un autre.

### 6.2.4 Génération du Diagramme de robustesse

Dans cette sous-section, nous allons expliquer les modifications apportées aux différentes étapes de la pipeline de génération du diagramme à partir des entités parsées. Les différentes étapes ont été décrites dans le chapitre précédent. Donc, nous ne citerons que celles qui ont subi des changements et nous terminerons par un diagramme du nouveau processus.

#### Création des objets du diagramme

Pour la création des DO, nous avons apporté quelques modifications simples. Premièrement, les mots composant la description de tous les objets à l'exception des *Control* sont transformés en leur lemme et tous les mots considérés comme Stop-words seront supprimés. Cela facilitera la lecture et la détection des objets partageant la même description.

Deuxièmement, un objet peut être de type *Entity* tout en étant une propriété. Cet ajout répond au besoin de la gestion du cas où un objet en compose un autre. Cette relation est identifiée par un *"of"* les liant *e.g., "data of site" "data"* et *"site"* sont des *Entity* mais l'objet *"data"* regroupe l'objet *"site"*, donc il doit être représenté comme une propriété. Nous avons dû opter pour cette solution car les *Property* ne sont pas gérées comme des *Entity* au travers du processus. Cette *Entity* perdra son statut de propriété si elle est fusionnée avec une autre qui n'est pas une propriété.

#### Déduction des interfaces

Comme nous l'avons expliqué précédemment, un grand nombre d'US ne sont pas munies d'une interface. Nous la déduisons donc sur base des *Entity* liées au *Control* qui n'en est pas muni. Certains d'entre eux ne possèdent pas d'enfant adéquat pour la déduction d'une interface. Nous avons pu observer que pour la plupart d'entre eux, ce sont des *Control* faisant référence à d'autres *Control*.

*e.g., " I want to be able to use a profile"*

Suite à cette observation, nous avons décidé que si un *Control* n'a pas d'*Entity* directement dans ses enfants, il utiliserait celles de ses enfants de type *Control* pour déduire son interface.

#### Fusion des objets similaires

Lors de la rédaction d'un backlog, il arrive que pour un même concept plusieurs termes soient employés. Dans notre modélisation, nous voulons éviter la création de plusieurs objets pour un même concept causé par l'emploi de différents noms. Pour cela, nous avons tenté d'identifier les objets similaires. D'abord, nous avons défini que pour que deux objets soient similaires, l'ensemble des mots qui composent la description de chacun d'eux doit trouver un mot similaire

dans la description de l'autre. Cela évite la fusion de deux objets partageant un même mot tout en étant des concepts distincts. Pour détecter la similarité entre deux mots, nous nous sommes donc tournés vers les différentes techniques d'analyse de similarité présentées dans l'état de l'art (Section 1.4). Dans notre situation, nous voulons identifier les mots partageant un sens commun. Deux techniques sont possibles, la proximité vectorielle et l'utilisation d'une base de données de mots similaires.

Tout d'abord, nous avons testé la proximité vectorielle à l'aide de SpaCy. En explorant les résultats, nous avons pu observer que certains mots qui ne devraient pas être proches le sont et vice versa. Nous pouvons expliquer cela par le vocabulaire plus spécifique employé et peut-être que l'entraînement d'un modèle sur base d'un ensemble de texte écrit par des développeurs pourrait améliorer nos résultats. Nous avons alors exploré une autre piste qui consiste à vérifier si un mot fait partie de la liste des synonymes d'un autre. Malheureusement, cette liste de synonymes est souvent trop large et selon le contexte, un mot n'est pas synonyme d'un autre.

Nous voulons éviter au maximum la fusion de deux objets s'ils ne sont pas des synonymes. Pour renforcer nos contraintes de décision, nous avons choisi de combiner les deux techniques citées précédemment. Leurs résultats combinés évitent le surplus de faux positif. Nous allons donc effectuer cette fusion des objets synonymes par type pour les *Actor*, *Entity* et *Property*. Cela n'est pas nécessaire pour les *Control* car nous avons constaté des fusions abusives causées par la faible différence entre certaines actions *e.g., "add a profile"* et *"create a profile"* qui seraient considérées comme similaires mais l'une des actions peut faire référence à l'ajout dans une liste et l'autre à la simple création. Un problème similaire a lieu pour les *Boundary*. Toutefois, nous allons fusionner celles ayant la même description pour éviter d'avoir des objets redondants.

## 6.3 Visualisation

Notre outil a la possibilité de générer trois types de diagramme de robustesse. Nous les présentons ci-dessous.

### 6.3.1 Diagramme de robustesse pour une histoire d'utilisateur seule

Le premier type crée un diagramme de robustesse permettant de visualiser une seule US. A la Figure 6.15, nous pouvons observer le diagramme de robustesse généré pour une US seule :

*"As a Publisher, I want to sign up for an account."*

Nous observons que l'*Actor "Publisher"* présent dans la *part-<acteur>* a été correctement modélisé. Le *Control "Sign up for an account"* a bien été mis en relation avec l'*Entity "Account"* et une *Boundary "Account Interface"* a été inférée pour permettre de relier l'*Actor* au *Control*.

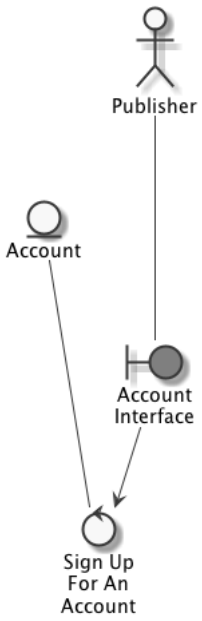


Figure 6.15: Exemple de diagramme de robustesse généré pour une histoire d'utilisateur

### 6.3.2 Diagramme de robustesse pour plusieurs histoires d'utilisateur

Notre outil est capable de fusionner les DR d'un ensemble d'US, ce qui permettra la génération d'un diagramme de robustesse unifié. En modélisant un ensemble d'US en un seul et même DR, nous pourrons aisément avoir une vue d'ensemble et facilement identifier les différents éléments/relations partagés entre celles-ci.

Par exemple à la Figue 6.16, nous avons combiné le DR de 5 US provenant du backlog Datahub (Table 6.1). Nous pouvons constater que l'ensemble des US sont bien modélisées sur un même diagramme unifié. L'*Entity "Data Package"* est modélisée une seule fois et reliée aux différents *Control* l'influençant (*"Delete", "Unpublish" ,"Import"*). Nous pouvons aussi constater qu'une *Boundary* unique *"Package Interface"* a été créée pour deux des trois *Control* affectant *"Data Package"*, une seconde *Boundary "Registry Interface"* a été inférée pour le troisième ("Import Data Package into the Registry"), car l'action n'influence par seulement l'*Entity "Data Package"*, mais aussi l'*Entity "Registry"*. On peut voir que la relation introduite par l'*"Into"* entre ces deux *Entity* est modélisée par une dépendance.

Table 6.1: Echantillon d'histoires d'utilisateur du backlog Datahub

| | **Histoire d'utilisateur** |
|---|---|
| 1 | As a Publisher, I want to permanently delete a data package. |
| 2 | As a Publisher, I want to unpublish a data package. |
| 3 | As a Publisher, I want to configure my client. |
| 4 | As a Publisher, I want to import my data package into the registry. |
| 5 | As a Publisher, I want to sign up for an account. |

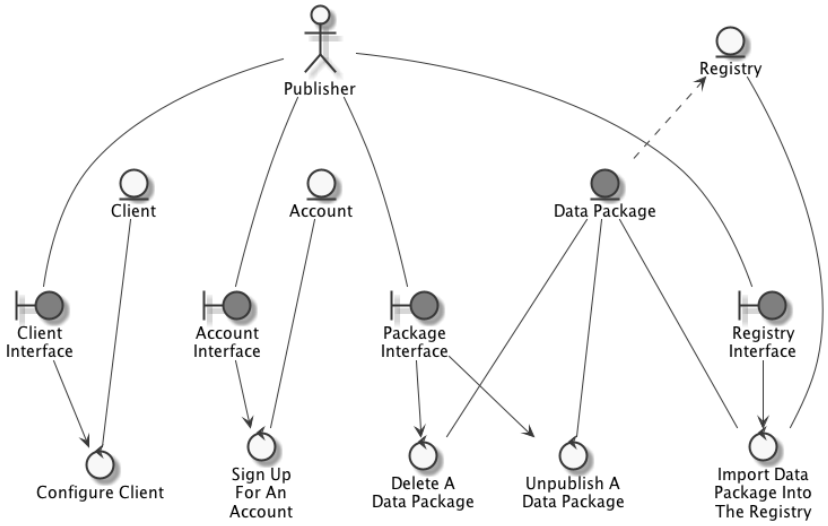


Figure 6.16: Diagramme de robustesse unifié depuis plusieurs histoires d'utilisateur

Prenons un second exemple qui fusionne le DR de 4US provenant du backlog Recycling (Table 6.2) à la Figure 6.17.

Table 6.2: Echantillon d'histoires d'utilisateur du backlog Recycling

| | **Histoire d'utilisateur** |
|---|---|
| 1 | As a Plan Review Staff member, I want to successfully Conduct a Plan Review Meeting with the Applicant and record the outcome. |
| 2 | As a Plan Review Staff member, I want to Track the Completion of Required Plan Reviews. |
| 3 | As an Inspection Staff member, I want to Create an Inspection. |
| 4 | As an Inspection Staff Supervisor, I want to Assign Inspections. |

Nous pouvons constater que les deux US ayant comme *Actor "Plan Review Staff member"* ont été correctement fusionnées. Les deux US parlant d'une *Entity "Inspection"* l'ont été également. On peut voir un exemple de relation de composition entre *"Completion"* qui est une *Property* et *"Plan Reviews"* qui est l'*Entity* dont elle fait partie. Un exemple de relation de dépendance est aussi présent entre les *Entity "Plan Review Meeting"* et *"Applicant"*. Nous pouvons aussi constater que

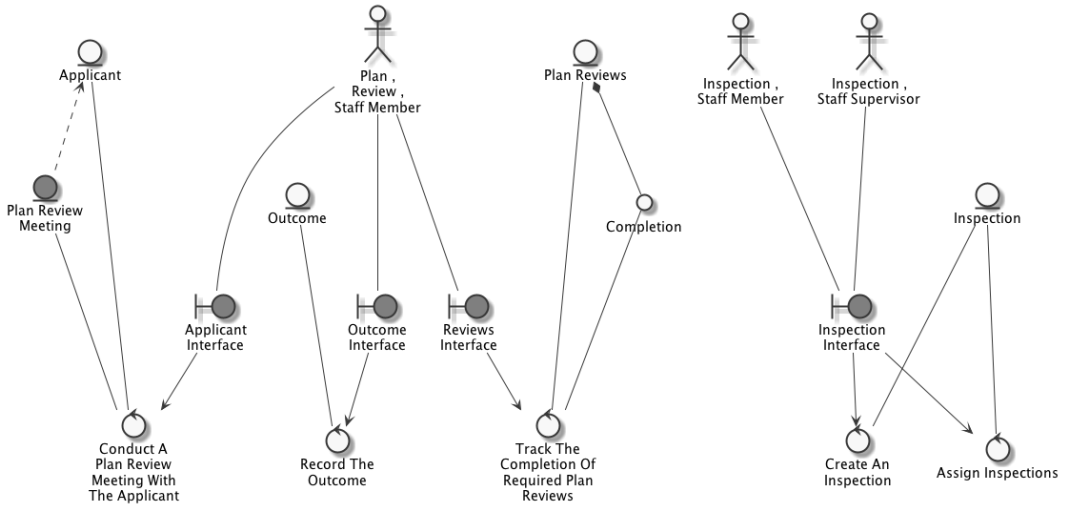


Figure 6.17: Diagramme de robustesse unifié depuis plusieurs histoires d'utilisateur composé de deux sous-diagrammes

notre outil a été capable de différencier *"Plan Reviews"* de *"Plan Review Meeting"* en créant deux *Entity* différentes. Cette fusion a permis de mettre en avant le lien partagé entre ces deux US tout en inférant une interface commune pour leurs deux *Actor*.

De plus, nous pouvons observer que ce second exemple est composé de deux sous-diagrammes, ce qui, d'après nos différentes expérimentations, n'est pas rare lors de la génération du DR d'un backlog entier. Un sous-diagramme est un diagramme de robustesse étant au moins composé d'un *Actor*, une *Boundary*, un *Control* et une *Entity* et n'étant pas relié à d'autres éléments. Lorsque le nombre d'US est trop important, il n'est pas rare d'obtenir un DR unifié illisible vu un trop grand nombre d'éléments. Pour résoudre ce problème, nous avons pensé à le découper en l'ensemble de ses sous-diagrammes valides, ce qui rendra la lecture de chacun beaucoup plus simple.

### 6.3.3 Diagramme de robustesse simplifié basé sur un *point-de-vue*

Nous proposons cette dernière manière de visualiser un backlog une fois encore pour faciliter la lecture d'un DR unifié trop important. Pour celle-ci, nous avons pensé à générer un DR correspondant au *"Point-De-Vue"* d'un élément. Nous proposons à notre utilisateur de pouvoir choisir un élément du DR unifié (*Actor*,*Boundary* et *Entity*) et un DR limité aux éléments en lien avec celui sélectionné sera généré. Par exemple à la Figure 6.18, nous pouvons voir le *"Point-De-Vue"* relatif au *Boundary* *"Interface Inspection"*. Nous observons aisément que les *Actor* peuvent interagir avec la *Boundary* sélectionnée et que les *Control* peuvent être effectués depuis celle-ci. Nous pouvons observer que les *Actor "Inspection Staff member"* et *" Inspection Staff Supervisor"* peuvent l'utiliser mais chacun d'eux ne peut pas accomplir les mêmes *Control* depuis celle-ci, selon les US. Cela permettra à l'architecte logiciel de s'interroger sur le fait de savoir si oui ou non une interface commune est une

bonne solution et comment identifier quelles actions sont possibles en fonction de l'utilisateur.

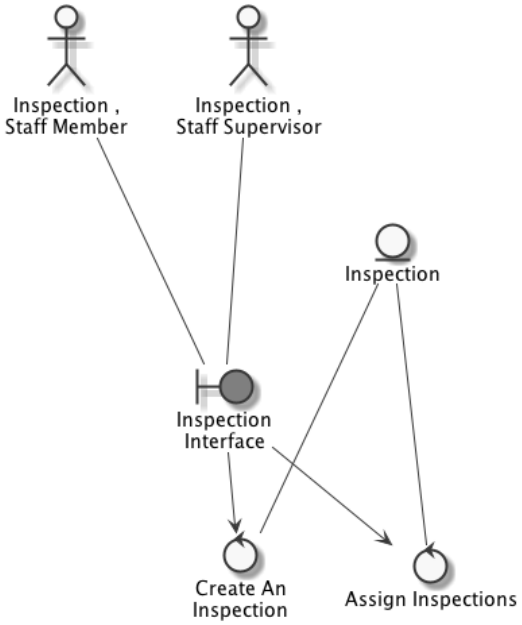


Figure 6.18: *"Point-De-Vue"* depuis la *Boundary* " Inspection Staff member"

Ce système de *"Point-De-Vue"* a pour objectif de permettre à un architecte logiciel de sélectionner un élément du système et de comprendre l'ensemble des relations/interactions qu'il peut avoir avec les autres tout en évitant que l'entièreté du système soit modélisé. Il pourra aussi accéder à la liste des US référençant au moins un des éléments modélisés dans ce *"Point-De-Vue"*. Cela lui permettra d'analyser la liste restreinte des US pouvant impacter cet élément du système.

Ci-dessous, nous expliquons en détail comment les différents éléments modélisés sont choisis en fonction du type de l'élément sélectionné.

#### *"Point-De-Vue"* depuis un *Actor*

Pour un *Actor*, nous voulons obtenir sa vision. Il nous faudra donc modéliser l'ensemble des *Boundary* avec lesquelles il interagit et l'ensemble des *Control* liés à celles-ci. Nous ajouterons l'ensemble des *Entity* et les *Property* de celles-ci. Avec ce *"Point-De-Vue"*, il deviendra beaucoup plus simple à l'équipe de développement d'identifier et d'analyser les différentes actions possibles sur le système pour un type d'utilisateur.

#### *"Point-De-Vue"* depuis une *Boundary*

Pour une *Boundary*, nous voulons visualiser l'ensemble des actions possibles depuis celle-ci et voir qui peut l'utiliser. D'abord, nous allons donc modéliser l'ensemble des *Actor* pouvant interagir

avec cette interface. Ensuite, nous ajouterons les différentes actions possibles (*Control*) depuis cette *Boundary* et les parties du système (*Entity*) affectées par celle-ci. Cette visualisation permettra de comprendre que certaines *Boundary* sont peut-être mal définies et qu'elles nécessitent d'être décomposées parce qu'elles permettent un trop grand nombre d'actions. Les développeurs pourront aussi observer les *Boundary* partagées par différents utilisateurs et mieux comprendre les fonctionnalités accessibles par chacun.

#### *"Point-De-Vue"* depuis une *Entity*

Pour une *Entity*, nous avons besoin de comprendre les différentes parties du système pouvant l'affecter. Nous allons donc d'abord modéliser l'ensemble des *Entity* et *Property* qui y sont liées. Ensuite, nous ajouterons tous les *Control* ayant accès à cette Entity et les *Boundary* permettant d'atteindre ces actions. Finalement, nous ajouterons tous les *Actor* qui peuvent utiliser ces *Boundary*. Cette dernière visualisation permettra d'identifier tous les comportements possibles vis-à-vis d'une *Entity* du domaine, *e.g.*, manque-t-il un type d'action, n'y a-t-il pas de redondance d'accès (sémantique identique non détectée automatiquement), ...

## 6.4 Evaluation

### 6.4.1 Questions de recherche

Pour pouvoir évaluer notre approche, nous avons réutilisé une des questions de recherche telles que définies par Gilson et al. [Gilson et al., 2019b] :

- **QR1: Les diagrammes de robustesse sont-ils automatiquement générés à partir d'histoires d'utilisateur syntaxiquement valides ?** Cette question de recherche vise à évaluer dans quelle mesure un processus automatique basé sur le traitement du langage naturel crée des diagrammes de robustesse syntaxiquement valides à partir d'histoires d'utilisateur réelles.

### 6.4.2 Méthode d'évaluation de notre question de recherche

Pour répondre à notre question de recherche, nous utiliserons le même dataset que celui utilisé au Chapitre 4. Pour rappel, celui-ci est composé de 1675 US réparties entre 22 backlogs provenant du monde industriel et universitaire.

Pour répondre à la QR1, nous allons évaluer si nos DR générés sont syntaxiquement corrects. Pour cela, nous avons défini les métriques suivantes :

- **Valid** : Tous les éléments dans DR sont valides, complets et connectés;
- **Invalid** : Quelques ou tous les éléments dans le DR ne sont pas connectés aux autres éléments.
- **Error** : Aucun DR n'a pu être généré (*e.g.*, parce que l'US ne respectait pas totalement le template de Cohn).

### 6.4.3 Résultats

A la Table 6.3, nous pouvons retrouver les résultats pour la QR1. D'abord, nous pouvons constater que les nombres d'US pour lesquelles aucun DR n'est généré sont presque nuls. Si nous analysons les deux US du backlog FrictionLess, nous pouvons observer que celles-ci n'utilisent pas le template de Cohn mais une variante incompatible avec notre outil (*e.g.,* des détails sont ajoutés entre la *part-<acteur>* et la *part-<objectif>*). Nous observons que c'est le backlog NSF qui possède la plus grande proportion de diagrammes invalides.

Table 6.3: Corpus des backlogs, histoires d'utilisateur et résultats pour QR1 [Gilson et al., 2019b]

| Backlog | Valid | Invalid | Error | Stories |
|---|---|---|---|---|
| FederalSpending | 72 | 22 | 0 | 94 |
| Loudoun | 53 | 4 | 0 | 57 |
| Recycling | 46 | 4 | 0 | 50 |
| OpenSpending | 49 | 4 | 0 | 53 |
| FrictionLess | 57 | 7 | 2 | 66 |
| ScrumAlliance | 84 | 13 | 0 | 97 |
| NSF | 53 | 19 | 0 | 72 |
| CamperPlus | 49 | 4 | 0 | 53 |
| PlanningPoker | 48 | 4 | 0 | 52 |
| DataHub | 60 | 7 | 0 | 67 |
| MIS | 66 | 16 | 1 | 83 |
| CASK | 50 | 14 | 0 | 64 |
| NeuroHub | 90 | 12 | 0 | 102 |
| Alfred | 117 | 17 | 0 | 134 |
| BadCamp | 62 | 7 | 0 | 69 |
| RDA-DMP | 54 | 28 | 0 | 82 |
| ArchiveSpace | 46 | 9 | 0 | 55 |
| UniBath | 51 | 2 | 0 | 53 |
| DuraSpace | 80 | 20 | 0 | 100 |
| RacDam | 94 | 6 | 0 | 100 |
| CulRepo | 88 | 26 | 0 | 114 |
| Zooniverse | 50 | 10 | 0 | 60 |

## 6.5 Discussions

### 6.5.1 Limitations techniques

Notre prototype actuel possède certaines limitations. Premièrement, nous avons pris la décision d'accepter toutes les US respectant le template de Cohn. Nous ne vérifions pas la qualité des US soumises. Celles ne respectant pas le template Cohn ou ne respectant pas assez les Frameworks permettant d'écrire des US de qualité génèrent souvent des DR totalement erronés. Pour résoudre ce problème, nous pourrions envisager l'utilisation d'un outil comme AQUSA [Lucassen et al.,

2016a] permettant d'identifier le non-respect des règles de qualité et proposant des alternatives d'écriture.

Deuxièmement, comme nous l'avons déjà dit, la génération de l'arbre des dépendances et du POS/TAG par SpaCy n'est pas exempte d'erreurs. Par le préprocessing, nous avons essayé de réduire ces cas, mais certains sont inévitables *e.g.*, un nom identifié comme verbe sera modélisé comme un *Control* à la place d'une *Entity*.

Troisièmement, la fusion des éléments en fonction de leur similarité peut être soumise à des erreurs, car celle-ci dépend d'une liste de mots similaires et de proximité vectorielle. Ces deux techniques ont été créées sur base de données générales, ce qui cause des identifications de similarité entre des mots spécifiques à un domaine, mais très proches dans un contexte général.

Quatrièmement, nous sommes limités par le nombre d'éléments modélisables laissant le diagramme compréhensible. Un trop grand nombre d'US entraîne la génération d'un DR beaucoup trop grand et illisible. Les approches par *"Point-De-Vue"* ou la découpe en sous-diagrammes tentent de résoudre ce problème mais restent inefficaces dans certaines situations où des éléments du système sont centraux et sont reliés à une trop grande partie du système.

Finalement, notre parseur n'est pas parfait. Voici quelques problèmes de modélisation identifiés :

- lorsqu'un ClausalComplement introduit par un adverbes comme *"where"* , *"what"* est présent dans une US, aucune relation ne sera modélisée entre les différents objets de son diagramme de robustesse.

  *e.g., "As a designer, I want to know* ***what early indications of hypotheses might be.****"*

- pour la première classe de PrepositionComplement (Section 6.2.3), notre token correspond à une action à réaliser avant l'action précédemment citée. Le problème de cette décision est que nous ne prenons pas en compte les cas où il s'agit d'une action qui ne doit pas être accomplie. Certaines prépositions introduisent une négation signifiant que pour réaliser l'action X, il faut que l'action Y n'ait pas eu lieu.

  *e.g., "As a Broker user, I want to submit records for individual recipients* ***without receiving a DUNS error.****"*

  Actuellement, nous n'avons pas choisi de moyen pour différencier ce cas de celui qui est positif.

- la négation d'une action n'est pas modélisée. Comme nous l'avons signalé dans le point précédent, nous n'avons pas de moyen pour modéliser la négation appliquée à une action. Actuellement, toutes les actions le sont comme positives, ce qui pourrait poser problème aux développeurs.

### 6.5.2 Faiblesses de notre technique d'évaluation

Dans la section 6.5, nous avons présenté la façon dont nous avons évalué notre technique. Malheureusement, celle-ci est soumise à plusieurs faiblesses. La première est la validation face au monde extérieur. Notre évaluation a été effectuée sur un nombre limité d'US provenant d'un OpenData. Il est donc évident que cet échantillon n'est pas représentatif de la totalité des US

présentées dans le monde réel. Comme nous l'avons déjà mentionné au Chapitre 4, il est très compliqué d'identifier une base de données couvrant l'ensemble des possibilités. Il est donc difficile de généraliser nos observations.

La seconde faiblesse porte sur le peu de paramètres couverts par notre évaluation. Actuellement, nous nous limitons à évaluer automatiquement la validité syntaxique de nos DR générés pour des US seules, ce qui est insuffisant. Le papier écrit par F. Gilson et al [Gilson et al., 2019b] présente différentes manières d'évaluer la qualité des diagrammes de robustesse générés par notre prototype. D'abord, il propose d'étudier la qualité sémantique en comparant des diagrammes de robustesse générés manuellement et automatiquement. Ensuite, il analyse la bonne unification des diagrammes de robustesse en comptant le nombre d'objets mal fusionnés. Enfin, il analyse les *"Points-De-Vue"* en vérifiant que tous les éléments modélisés sur chacun d'eux influencent l'élément sur lequel se base le *"Point-De-Vue"*.

CHAPTER 7

# CONCLUSION ET TRAVAUX FUTURS

*Ce chapitre conclut notre mémoire et nous présenterons nos travaux futurs.*

## 7.1 Conclusion

En développement logiciel Agile, la tâche d'analyse et de maintien d'un backlog n'est pas aisée, particulièrement quand le nombre d'histoires d'utilisateur est élevé. Sur base de cette analyse, les architectes logiciels devront comprendre le domaine du projet et identifier les histoires d'utilisateur importantes pour leur prise de décisions de design.

Dans nos recherches, nous avons tenté d'apporter deux moyens permettant de les aider dans leur travail grâce à des solutions automatisées s'appuyant sur l'analyse du langage naturel et l'apprentissage automatisé. Afin de concevoir et d'expérimenter nos techniques, nous avons utilisé une base de données OpenSource composée de 1,675 histoires d'utilisateur réparties entre 22 backlogs de projets industriels et universitaires. Deux experts ont analysé manuellement l'ensemble des histoires d'utilisateur afin d'identifier les attributs de qualité auxquels chacune d'elles faisait référence.

Premièrement, les attributs de qualité étant un des facteurs clés influençant les décisions de design, nous proposons une approche permettant de détecter les histoires d'utilisateur faisant référence à l'un d'eux et d'identifier leur type à l'aide de modèles de classification d'apprentissage automatisé. Nos résultats prouvent la viabilité de notre idée avec un *F1-Score* de 0.71 pour la détection des histoires d'utilisateur liées à un attribut de qualité pour l'encodage TF-IDF utilisé avec un modèle ComplementNB et un *F1-Score Global* de 0.53 pour les identifier à l'aide d'un modèle SpaCy (avec comme un meilleur *F1-Score* de 0.63 pour *compatibility*). Toutefois, ces résultats restent mitigés, mais nous avons identifié différentes manières d'optimiser nos approches.

Nous avons aussi observé que certains attributs de qualité comme *usability* seront beaucoup plus complexes à identifier de par la grande variété des aspects couverts par celui-ci. De plus, nous proposons un premier classement naïf des attributs de qualité les plus importants basés uniquement sur le nombre d'histoires d'utilisateur identifiées comme faisant référence à un type d'attribut de qualité. Nous pensons que celui-ci n'est pas suffisant et qu'il faudrait prendre en compte la valeur business de chaque histoire d'utilisateur identifiée dans le calcul de l'importance.

Deuxièmement, nous avons utilisé le diagramme de robustesse afin de modéliser les histoires d'utilisateur contenues dans un backlog. Pour cela, nous avons développé une pipeline composée d'un module de Préprocessing, un parseur utilisant le Part-of-Speech et l'arbre des dépendances généré par SpaCy ainsi qu'un module créant le code PlantUML permettant de générer les diagrammes de robustesse. Le module de Préprocessing servira à réduire la complexité de l'histoire d'utilisateur afin de faciliter le travail de SpaCy et celui de notre parseur. Le parseur, à l'aide des règles que nous avons créées par une approche inductive, traversera l'arbre des dépendances de l'histoire d'utilisateur pour extraire les différents candidats. Le dernier module fusionne tous les candidats similaires et générera le code PlantUML du diagramme de robustesse lui correspondant.

Nous proposons trois types de visualisation à l'aide de diagrammes de robustesse. La première permet de visualiser une histoire d'utilisateur seule. La seconde unifie les diagrammes de robustesse de plusieurs d'entre elles. La dernière vient répondre au problème de surinformation sur les diagrammes de robustesse unifiés en proposant des *"Points-De-Vue"* centrés autour d'un élément du diagramme de robustesse. Celle-ci consiste à afficher uniquement les objets en interagissant avec notre sélection. Après une simple évaluation de la syntaxe, nous pensons être sur la bonne voie, mais d'autres tests sont nécessaires.

## 7.2 Travaux futurs

### 7.2.1 Agrandissement de la base de données

Un des objectifs majeurs consistera à augmenter le nombre d'histoires d'utilisateur composant notre base de données à l'aide de nouveaux backlogs issus du monde industriel. Ceux-ci devraient, de préférence, être écrits par d'autres équipes de développement pour permettre de découvrir de nouvelles habitudes utilisées pour rédiger une histoire d'utilisateur. Ces nouveaux exemples permettront de tester notre générateur de diagrammes de robustesse et peut-être de trouver de nouveaux types de cas encore mal générés.

Ces nouvelles données seront d'autant plus précieuses pour l'entraînement de nos modèles de prédiction d'attributs de qualité liés à une histoire d'utilisateur. Comme nous avons pu le montrer, nos classifieurs peuvent encore améliorer leurs performances à l'aide de nouvelles données d'entraînement. Si nous voulons pouvoir détecter efficacement l'ensemble des types d'attribut de qualité, il est très important de trouver un maximum d'histoires d'utilisateur liées aux attributs de qualité les moins représentés dans nos données *e.g., reliability* avec ses 28 occurrences a été difficilement identifié par l'ensemble des modèles testés. Toutefois, nous avons pu observer une exception pour *usability* qui possède la troisième plus grande quantité d'exemples, mais qui est

pourtant l'un des attributs de qualité les moins bien identifiés. Nous expliquons cela par le manque de vocabulaire spécifique à *usability*.

### 7.2.2 Identification des attributs de qualité présents dans une histoire d'utilisateur

Comme nous l'avons expliqué, ces premières expérimentations n'avaient pas pour but de trouver la solution optimale à travers les possibilités d'apprentissage automatisé pour la détection et l'identification des attributs de qualité liés à une histoire d'utilisateur mais elles avaient pour but de montrer que cela était faisable. Ayant prouvé avec des modèles non optimaux la faisabilité de notre tâche, il est normal de prévoir l'expérimentation sur de meilleures solutions. Si nous voulons continuer à utiliser de l'apprentissage profond, il est judicieux de s'orienter vers l'état de l'art actuel, BERT. Les expérimentations des modèles plus simples (LinearSVC, ComplementNB, LogisticRegression) avec différents encodages ont pu montrer que celui proposé par BERT offrait à chaque fois de meilleurs résultats. Nous pensons donc que l'implémentation d'une pipeline totale avec BERT comme encodeur et classifieur pourra nous offrir de meilleurs résultats.

Si BERT n'obtient pas les résultats escomptés, nous pourrions nous tourner vers l'approche du One-Vs-Rest mais cette fois en utilisant le modèle le plus performant pour la détection de chaque attribut de qualité spécifique. Cette idée est née des résultats de nos travaux qui montrent que certains types de modèles étaient meilleurs que d'autres pour certains attributs de qualité. Si nous optons pour cette approche, il sera crucial de tester d'autres types de modèles et d'optimiser leurs méta-paramètres afin d'obtenir une solution optimale.

### 7.2.3 Modélisation de diagrammes de robustesse à partir d'histoires d'utilisateur

Comme nous l'avons expliqué précédemment, nous avons résolu une grande partie des problèmes et le nombre d'histoires d'utilisateur bien modélisées a largement augmenté. Malheureusement, certains cas n'ont pas pu être solutionnés par manque de temps. Pour faciliter l'analyse de ces derniers, une liste d'histoires d'utilisateur ne générant pas de diagramme de robustesse ou ayant des objets non connectés a été créée. Certains cas ont déjà pu être documentés. Il serait intéressant d'ajouter un module de détection de non-respect des Framework d'écriture d'une histoire d'utilisateur expliquant aux développeurs que leur manque de rigueur pourrait causer des erreurs de modélisation. L'intégration dans notre pipeline de AQUSA développé par Lucassen [Lucassen et al., 2016a], pourrait détecter ces mauvaises pratiques et proposer des alternatives qui amélioreraient la qualité des histoires d'utilisateur.

# BIBLIOGRAPHY


ACL (2019). Pos tagging (state of the art). https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art). Consulté le 25/03/2019.

Al Omran, F. N. A. and Treude, C. (2017). Choosing an nlp library for analyzing software documentation: A systematic literature review and a series of experiments. In **2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)**, pages 187–197.

Apache (2018a). Apache hadoop. http://hadoop.apache.org. Consulté le 10/05/2019.

Apache (2018b). Struts. https://struts.apache.org. Consulté le 10/05/2019.

Apache (2019). Spark. https://spark.apache.org/. Consulté le 10/05/2019.

Apple (2010). Ibook. https://www.apple.com/lae/apple-books/. Consulté le 10/05/2019.

Bashir, U. and Chachoo, M. (2017). Performance evaluation of j48 and bayes algorithms for intrusion detection system. **International Journal of Network Security and Its Applications**, 9:01–11.

Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. **Information and Software Technology**, 75:1 – 16.

Bass, L., Klein, M., and Bachmann, F. (2002). Quality attribute design primitives and the attribute driven design method. In van der Linden, F., editor, **Software Product-Family Engineering**, pages 169–186, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bertoa, M. F. and Vallecillo, A. (2002). Quality attributes for cots components. **Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)**, 1(2):128–144.

Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., and Matthes, F. (2017). Automatic extraction of design decisions from issue management systems: A machine learning based approach. In **Software Architecture**, pages 138–154, Cham. Springer Int'l Pub.

Bobriakov, I. (2018). Comparison of top 6 python nlp libraries. https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb. Consulté le 25/03/2019.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. **CoRR**, abs/1607.04606.

Borah, P., Talukdar, G., and Boruah, A. (2014). Approaches for word sense disambiguation - a survey. **International Journal of Recent Technology and Engineering**, 3:135–138.

Bornstein, A. A. (2018). Beyond word embeddings part 2. https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec. Consulté le 25/03/2019.

Caire, P., Genon, N., Heymans, P., and Moody, D. (2013). **Visual notation design 2.0: Towards user comprehensible requirements engineering notations**, pages 115–124.

Chang, C.-C. and Lin, C.-J. (2007). Libsvm: A library for support vector machines. **ACM Transactions on Intelligent Systems and Technology**, 2.

Clark, K. and Manning, C. D. (2016). Improving coreference resolution by learning entity-level distributed representations. **CoRR**, abs/1606.01323.

Cohn, M. (2004). **User Stories Applied: For Agile Software Development**. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Dalpiaz, F. (2018). Requirements data sets (user stories) Mendeley Data, v1. http://dx.doi.org/10.17632/7zbk8zsd8y.1.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. **CoRR**, abs/1810.04805.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. **Information and Software Technology**, 50(9-10):833 – 859.

El-Attar, M. and Miller, J. (2010). Developing comprehensive acceptance tests from use cases and robustness diagrams. **Requirements Engineering**, 15(3):285–306.

Elallaoui, M., Nafil, K., and Touahni, R. (2015). Automatic generation of uml sequence diagrams from user stories in scrum process. In **10th International Conference on Intelligent Systems: Theories and Applications**, pages 1–6.

Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into uml use case diagrams using nlp techniques. **Procedia Comput. Sci.**, 130(C):42–49.

Facebook (2014). Whatsapp. https://www.whatsapp.com. Consulté le 10/05/2019.

Feng, J., Xu, H., Mannor, S., and Yan, S. (2014). Robust logistic regression and classification. In **Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1**, NIPS'14, pages 253–261, Cambridge, MA, USA. MIT Press.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. **J. Mach. Learn. Res.**, 3:1289–1305.

Friedrich, F., Mendling, J., and Puhlmann, F. (2011). Process model generation from natural language text. In Mouratidis, H. and Rolland, C., editors, **Advanced Information Systems Engineering**, pages 482–496, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gacek, C., Abd-allah, A., Clark, B., and Boehm, B. (1995). On the definition of software system architecture. In **Proc. of ICSE 17 Software Architecture Workshop**.

Galster, M., Gilson, F., and Georis, F. (2019). Automatic generation and combination of visual use case scenarios from textual user stories. In **13th European Conference on Software Architecture (ECSA 2019)**.

Gandhi, R. (2018). Support vector machine : Introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Consulté le 25/03/2019.

Genon, N. (2016). **Visual Notation Design 2.0: Empowering End-Users for Semantically Transparent Notations**. PhD thesis, University of Namur.

Gilson, F. (2015). **Transformation-Wise Software Architecture Framework : A Transformational Approach to Design Component-Based Systems**. PhD thesis, University of Namur.

Gilson, F., Galster, M., and Georis, F. (2019a). Extracting quality attributes from user stories for early architecture decision making. In **IEEE International Conference on Software Architecture Workshops)**, pages 129–136. IEEE.

Gilson, F., Galster, M., and Georis, F. (2019b). Preliminary insights from automatically identifying quality attributes in user stories. In **The 34th IEEE/ACM International Conference on Automated Software Engineering (ASE 2019)**.

Gilson, F. and Irwin, C. (2018). From user stories to use case scenarios - towards a generative approach. In **Proc. of the 25th Australasian Software Engineering Conf.**, pages 61 – 65. IEEE Computer Society.

Gorschek, T., Tempero, E., and Angelis, L. (2014). On the use of software design models in software development practice: An empirical investigation. **Journal of Systems and Software**, 95:176 – 193.

Harel, D. and Rumpe, B. (2004). Meaningful modeling: what's the semantics of "semantics"? **Computer**, 37(10):64–72.

Hoda, R. and Murugesan, L. (2016). Multi-level agile project management challenges: A self-organizing team perspective. **Journal of Systems and Software**, 117:245–257.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2003). A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University.

HuggingFace (2019). Neuralcoref: Coreference resolution in spacy with neural networks. https://github.com/huggingface/neuralcoref. Consulté le 25/03/2019.

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., and Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. **Computers in Human Behavior**, 51:915 – 929. Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era.

ISO/IEC (2010). Iso/iec 25010 system and software quality models. Technical report, Internation Organization for Standardization/International Electrotechnical Commission.

Jacobson, I. (1987). Object-oriented development in an industrial environment. In **Conference Proceedings on Object-oriented Programming Systems, Languages and Applications**, OOPSLA '87, pages 183–191. ACM.

Jansen, A. (2008). **Architectural design decisions**. PhD thesis. Relation: https://www.rug.nl/date_submitted:2008 Rights: University of Groningen.

Jansen, A. and Bosch, J. (2005). Software architecture as a set of architectural design decisions. In **Proc. of the 5th Working Conf. on Software Architecture**, pages 109–120, Washington, DC, USA. IEEE Computer Society.

Kassab, M. (2014). An empirical study on the requirements engineering practices for agile software development. In **2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications**, pages 254–261.

Kent Beck, Mike Beedle, A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. D. T. (2001). Manifeste pour le développement agile de logiciels. http://agilemanifesto.org/iso/fr/manifesto.html. Consulté le 20/03/2019.

Kondrak, G. (2005). N-gram similarity and distance. In Consens, M. and Navarro, G., editors, **String Processing and Information Retrieval**, pages 115–126, Berlin, Heidelberg. Springer Berlin Heidelberg.

Lu, M. and Liang, P. (2017). Automatic classification of non-functional requirements from augmented app user reviews. In **Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering**, EASE'17, pages 344–353, New York, NY, USA. ACM.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2015). Forging high-quality user stories: Towards a discipline for agile requirements. In **2015 IEEE 23rd International Requirements Engineering Conference (RE)**, pages 126–135.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2016a). Improving agile requirements: the quality user story framework and tool. **Requirements Engineering**, 21(3):383–403.

Lucassen, G., Dalpiaz, F., Werf, J. M. E. M. v. d., and Brinkkemper, S. (2016b). The use and effectiveness of user stories in practice. In Daneva, M. and Pastor, O., editors, **Requirements Engineering: Foundation for Software Quality**, pages 205–222, Cham. Springer Int'l Pub.

Malik Hneif, S. H. O. (2009). Review of agile methodologies in software development. In **International Journal of Research and Reviews in Applied Sciences**.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013). Linguistic regularities in continuous space word representations. In **Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies**, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Moody, D. (2009). The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. **IEEE Transactions on Software Engineering**, 35(6):756–779.

NLP., S. (2019). Corenlp. https://github.com/stanfordnlp/CoreNLP. Consulté le 25/03/2019.

OMG (2011). Business Process Model and Notation (BPMN), Version 2.0.

Osman, C.-C. and Zalhan, P.-G. (2016). From Natural Language Text to Visual Models: A survey of Issues and Approaches. **Informatica Economica**, 20(4):44–61.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In **Empirical Methods in Natural Language Processing (EMNLP)**, pages 1532–1543.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. **SIGSOFT Software Engineering Notes**, 17(4):40–52.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018a). Deep contextualized word representations. **CoRR**, abs/1802.05365.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018b). Deep contextualized word representations. **CoRR**, abs/1802.05365.

Petre, M. (2013). Uml in practice. In **Proceedings of the International Conference on Software Engineering**, ICSE'13, pages 722–731, Piscataway, NJ, USA. IEEE Press.

Rennie, J. D. M., Shih, L., Teevan, J., and Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In **Proceedings of the Twentieth International Conference on International Conference on Machine Learning**, ICML'03, pages 616–623. AAAI Press.

Rong, X. (2014). word2vec parameter learning explained. **CoRR**, abs/1411.2738.

Rosenberg, D. and Stephens, M. (2007). **Use Case Driven Object Modeling with UML: Theory and Practice**. Apress, Berkely, CA, USA.

Royce, W. W. (1970). Managing the development of large software systems. In **Proc. of IEEE WESCON 26**, pages 1–9. IEEE Press.

Ruder, S. (2019). Dependency parsing. http://nlpprogress.com/english/dependency_parsing.html. Consulté le 25/03/2019.

Schön, E.-M., Thomaschewski, J., and Escalona, M. J. (2017). Agile requirements engineering: A systematic literature review. **Computer Standards & Interfaces**, 49:79 – 91.

Schön, E.-M., Winter, D., Escalona, M. J., and Thomaschewski, J. (2017). Key challenges in agile requirements engineering. In Baumeister, H., Lichter, H., and Riebisch, M., editors, **Agile Processes in Software Engineering and Extreme Programming**, pages 37–51, Cham. Springer Int'l Pub.

Schwaber, K. and Beedle, M. (2001). **Agile Software Development with Scrum**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Shahbazian, A., Lee, Y. K., Le, D., Brun, Y., and Medvidovic, N. (2018). Recovering architectural design decisions. In **2018 IEEE International Conference on Software Architecture (ICSA)**, pages 95–9509.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. **Science**, 362(6419):1140–1144.

Simon, H. A. (1996). **The Sciences of the Artificial (3rd Ed.)**. MIT Press, Cambridge, MA, USA.

Stavru, S. (2014). A critical examination of recent industrial surveys on agile method usage. **Journal of Systems and Software**, 94:87 – 97.

Store, A. (2018). String similarity algorithms compared. https://medium.com/@appaloosastore/string-similarity-algorithms-compared-3f7b4d12f0ff. Consulté le 25/03/2019.

Störrle, H. (2017). How are conceptual models used in industrial software development?: A descriptive survey. In **Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering**, EASE'17, pages 160–169, New York, NY, USA. ACM.

Tang, A., Razavian, M., Paech, B., and Hesse, T. (2017). Human aspects in software architecture decision making: A literature review. In **2017 IEEE International Conference on Software Architecture (ICSA)**, pages 107–116.

Tensorflow (2019). Vector representations of words. https://www.tensorflow.org/tutorials/representation/word2vec. Consulté le 25/03/2019.

Tran, C. T., Zhang, M., Andreae, P., and Xue, B. (2017). Bagging and feature selection for classification with incomplete data. pages 471–486.

Treude, C., Robillard, M. P., and Dagenais, B. (2015). Extracting development tasks to navigate software documentation. **IEEE Transactions on Software Engineering**, 41(6):565–581.

Vapnik, V. N. (1995). **The Nature of Statistical Learning Theory**. Springer-Verlag, Berlin, Heidelberg.

Verdi, M. P., Crooks, S. M., and White, D. R. (2002). Learning effects of print and digital geographic maps. **Journal of Research on Technology in Education**, 35(2):290–302.

Vliet, H. and Tang, A. (2016). Decision making in software architecture. **Journal of Systems and Software**, 117.

Wake, B. (2003). INVEST in good stories, and smart tasks. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Consulté le 12/09/2018.

Wautelet, Y., Heng, S., Kolp, M., and Mirbel, I. (2014a). Unifying and Extending User Story Models. In **25th International Conference on Advanced Information Systems Engineering**, Thessaloniki, Greece.

Wautelet, Y., Heng, S., Kolp, M., and Mirbel, I. (2014b). Unifying and Extending User Story Models. In **25th International Conference on Advanced Information Systems Engineering**, Thessaloniki, Greece.

Webster, J. J. and Kit, C. (1992). Tokenization as the initial phase in nlp. In **Proceedings of the 14th Conference on Computational Linguistics - Volume 4**, COLING '92, pages 1106–1110, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wikipedia (2019a). Activation function. https://en.wikipedia.org/wiki/Activation_function. Consulté le 10/05/2019.

Wikipedia (2019b). Cosine similarity. https://en.wikipedia.org/wiki/Cosine_similarity. Consulté le 10/05/2019.

Wikipedia (2019c). Naive bayes classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier. Consulté le 10/05/2019.

Wikipedia (2019d). Sensitivity and specificity. https://en.wikipedia.org/wiki/Sensitivity_and_specificity. Consulté le 25/03/2019.

Wolf, T. (2017). State-of-the-art neural coreference resolution for chatbots. https://medium.com/huggingface/state-of-the-art-neural-coreference-resolution-for-chatbots-3302365dcf30. Consulté le 25/03/2019.

Wu, H. C., Luk, R. W. P., Wong, K. F., and Kwok, K. L. (2008). Interpreting tf-idf term weights as making relevance decisions. **ACM Trans. Inf. Syst.**, 26(3):13:1–13:37.

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. **CoRR**, abs/1702.01923.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2017). Recent trends in deep learning based natural language processing. **CoRR**, abs/1708.02709.

Zhang, H. and Li, D. (2007). Naïve bayes text classifier. In **2007 IEEE International Conference on Granular Computing (GRC 2007)**, pages 708–708.

Zhang, Y., Jin, R., and Zhou, Z.-H. (2010). Understanding bag-of-words model: A statistical framework. **International Journal of Machine Learning and Cybernetics**, 1:43–52.

Zurcher, F. W. and Randell, B. (1968). Iterative multi-level modeling - a methodology for computer system design. In **Proc. of IFIP World Computer Congress**, pages 867–871. IEEE CS Press.